

MATLAB 仿真应用精品丛书

MATLAB R2016a

神经网络设计应用 27 例

顾艳春 编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书以 MATLAB R2016a 为平台,通过专业技术与大量典型实例相结合,介绍了各种典型网络的训练过程和实际应用。全书共 27 个案例,从实用角度出发,详尽地讲述感知器网络、线性神经网络、RBF 神经网络、BP 神经网络、反馈神经网络及自组织神经网络等内容,扩展介绍神经网络在其他工程领域的实际应用。

本书可作为科研人员及工程技术人员的参考用书,也可作为本科生和研究生的学习用书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

MATLAB R2016a 神经网络设计应用 27 例/顾艳春编著. —北京:电子工业出版社, 2018.1
(MATLAB 仿真应用精品丛书)
ISBN 978-7-121-33329-3

I. ①M… II. ①顾… III. ①Matlab 软件—应用—神经网络 IV. ①TP183

中国版本图书馆 CIP 数据核字(2017)第 316127 号

策划编辑:陈韦凯

责任编辑:万子芬 特约编辑:徐 宏

印 刷:

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1092 1/16 印张:29.25 字数:749 千字

版 次:2018 年 1 月第 1 版

印 次:2018 年 1 月第 1 次印刷

印 数:2 500 册 定价:69.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式: chenwk@phei.com.cn。

前 言

1943 年, 心理学家 W.S.McCulloch 和数理逻辑学家 W.Pitts 建立了神经网络和数学模型, 称为 MP 模型。他们通过 MP 模型提出了神经元的形式化数学描述和网络结构方法, 证明了单个神经元能执行逻辑功能, 从而开创了人工神经网络研究的时代。近年来, 人工神经网络正在模拟人类认知的道路上更加深入发展, 与模糊系统、遗传算法、进化机制等结合, 形成计算智能, 成为人工智能的一个重要方向, 将在实际应用中得到发展。将信息几何应用于人工神经网络的研究, 为人工神经网络的理论研究开辟了新的途径。神经计算机的研究发展很快, 已有产品进入市场。光电结合的神经计算机为人工神经网络的发展提供了良好条件。

神经网络是通过对人脑的基本单元——神经元的建模和连接, 探索模拟人脑神经系统功能的模型, 并研制一种具有学习、联想、记忆和模式识别等智能信息处理功能的人工系统。神经网络的一个重要特性是它能够从环境中学习, 并把学习的结果分布存储于网络的突触连接中。神经网络的学习是一个过程, 在其所处环境的激励下, 相继给网络输入一些样本模式, 并按照一定的规则(学习算法)调整网络各层的权值矩阵, 待网络各层权值都收敛到一定值, 学习过程结束, 之后就可以用生成的神经网络来对真实数据分类。

神经网络在很多领域中已得到了很好的应用, 但其需要研究的方面还很多。其中, 具有分布存储、并行处理、自学习、自组织及非线性映射等优点的神经网络与其他技术的结合, 以及由此而来的混合方法和混合系统, 已经成为一大研究热点。由于其他方法也有它们各自的优点, 所以将神经网络与其他方法相结合, 取长补短, 继而可以获得更好的应用效果。目前, 这方面的工作有神经网络与模糊逻辑、专家系统、遗传算法、小波分析、混沌、粗集理论、分形理论、证据理论和灰色系统等融合。

MATLAB 自产生之日起就具有方便的数据可视化功能, 以将向量和矩阵用图形表现出来, 并且可以对图形进行标注和打印。高层次的作图包括二维和三维的可视化、图像处理、动画和表达式作图, 可用于科学计算和工程绘图。新版本的 **MATLAB** 对整个图形处理功能进行了很大的改进和完善, 使它不仅在一般数据可视化软件都具有的功能(例如二维曲线和三维曲面的绘制和处理等)方面更加完善, 而且一些其他软件所没有的功能(例如图形的光照处理、色度处理以及四维数据的表现等), **MATLAB** 同样表现了出色的处理能力。同时, 对一些特殊的可视化要求(例如图形对话等), **MATLAB** 也有相应的功能函数, 保证了用户不同层次的要求。另外, 新版本的 **MATLAB** 还着重在图形用户界面(GUI)的制作上做了很大的改进, 可以满足对这方面有特殊要求的用户的需求。

MATLAB 对许多专门的领域都开发了功能强大的模块集和工具箱。一般来说, 它们都是由特定领域的专家开发的, 用户可以直接使用工具箱学习、应用和评估不同的方法而不需要自己编写代码。**MATLAB** 在很多领域, 如数据采集、数据库接口、概率统计、样条拟合、优化算法、偏微分方程求解、神经网络、小波分析、信号处理、图像处理、系统辨识、控制系统设计、LMI 控制、鲁棒控制、模型预测、模糊逻辑、金融分析、地图工具、非线性控制设计、实时快速原型及半物理仿真、嵌入式系统开发、定点仿真、DSP 与通信、电力系统仿真等, 都在工具箱(Toolbox)家族中有了自己的一席之地。

一种语言之所以能够如此迅速地普及和应用, 显示出如此旺盛的生命力, 是因为它有着

不同其他语言的特点。正如 C 语言等高级语言使人们摆脱了需要直接对计算机硬件资源进行操作的要求，被称为第四代计算机语言的 MATLAB（简称 M 语言），利用其丰富的函数资源和工具箱资源，使编程人员可以根据不同的需要选择相应的优化函数，而不需要编写烦琐的程序代码。该软件最突出的特点就是简洁、开放式、便捷等，它提供了更为直观、符合人们思维习惯的代码。同时给用户带来最直观、最简洁的程序开发环境。目前的 MATLAB 可以说是科技工作者必不可少的工具之一，掌握这一重要工具将使得日常的学习和工作事半功倍。

MATLAB 之所以有如此强大的功能在于其还在不断扩大的工具箱的应用，离开了工具箱的应用，MATLAB 环境下的操作也仅仅是简单的矩阵运算与作图而已。神经网络工具箱正是在 MATLAB 环境下所开发出来的众多工具箱之一，它是以人工神经网络理论为基础，用 MATLAB 语言构造出典型神经网络的激活函数，根据各种典型的修正网络权值的规则，加上网络的训练过程，用 MATLAB 编写出各种网络权值训练的子程序，网络的设计者可以根据所需去调用工具箱中有关神经网络的设计与训练程序，使自己能够从烦琐的编程中解脱出来，致力于思考问题和解决问题，从而提高效率和解题质量。

本书基于 MATLAB R2016a 全面讲解 MATLAB 相关知识，帮助读者尽快掌握 MATLAB 的应用。本书具有如下特点：

（1）全面细致，循序渐进。本书以 MATLAB R2016a 为平台，简要、全面、由浅入深地介绍 MATLAB 软件的特色、使用，再辅以 MATLAB 在工程中的应用案例，帮助读者尽快掌握用 MATLAB 进行工程应用分析的技能。

（2）内容新颖，应用典型。本书结合 MATLAB 解决工程应用中的各种实际问题，详细地讲解 MATLAB 软件的使用方法与技巧，并通过大量典型的应用例子来实操，在讲解过程中辅以相应的图片，使读者在阅读时一目了然，从而快速掌握书中的内容。

（3）轻松易学，上手快速。本书理论与实例相结合，并通过 MATLAB 的在线帮助、自带实例等内容，使读者轻松掌握所学内容，快速上手，还可以提高快速分析和解决实际问题的能力，从而能够在最短的时间内，以最高的效率解决实际通信系统中遇到的问题。

本书主要由顾艳春编写，参加编写的还有赵书兰、刘志为、栾颖、王宇华、吴茂、方清城、邓奋发、何正风、丁伟雄、李娅、辛焕平、杨文茵、李晓东和张德丰。

本书可以作为广大科研人员、学者、工程技术人员的参考用书，也可以作为广大在校本科生和研究生的学习用书。本书提供案例源代码下载，读者可以登录华信教育资源网（www.hxedu.com.cn）查找本书，免费下载。

由于时间仓促，加之作者水平有限，所以错误和疏漏之处在所难免。在此，诚恳地期望得到各领域的专家和广大读者的批评指正。

编 著 者

目 录

第 1 章	RBF 神经网络的实际应用	1
1.1	用于曲线拟合的 RBF 神经网络	1
1.2	径向基网络实现非线性函数回归	10
1.3	CRNN 网络应用	13
1.4	PNN 网络应用	15
1.5	RBF 神经网络的优缺点	19
第 2 章	SOM 网络算法分析与应用	22
2.1	SOM 网络的生物学基础	22
2.2	SOM 网络的拓扑结构	22
2.3	SOM 网络的权值调整	23
2.4	SOM 网络的 MATLAB 实现	26
2.5	SOM 网络的应用	33
第 3 章	线性网络的实际应用	45
3.1	线性化建模	45
3.2	模式分类	50
3.3	消噪处理	51
3.4	系统辨识	54
3.5	系统预测	55
第 4 章	BP 网络算法分析与应用	61
4.1	BP 网络模型	61
4.2	BP 网络学习算法	62
4.2.1	BP 网络学习算法介绍	62
4.2.2	BP 网络学习算法的比较	67
4.3	BP 神经网络特点	68
4.4	BP 网络功能	68
4.5	BP 网络实例分析	68
第 5 章	神经网络在选址与地震预测中的应用	78
5.1	配送中心选址	78
5.2	地震预报	81
5.2.1	问题概述	82
5.2.2	网络设计	83
5.2.3	网络训练与测试	83
5.2.4	网络实现	88
第 6 章	模糊神经网络的算法分析与实现	91
6.1	模糊神经网络的形式	91





6.2	神经网络和模糊控制结合的优点	92
6.3	神经模糊控制器	92
6.4	神经模糊控制器的学习算法	95
6.5	模糊神经网络 MATLAB 函数	97
6.5.1	模糊神经系统的建模函数	97
6.5.2	采用网格分割方式生成模糊推理系统函数	102
6.6	MATLAB 模糊神经推理系统的图形用户界面	103
第 7 章	BP 网络的典型应用	107
7.1	数据归一化方法	107
7.2	提前终止法	109
7.3	BP 网络的局限性	111
7.4	BP 网络典型应用	112
7.4.1	用 BP 网络估计胆固醇含量	112
7.4.2	线性神经网络在信号预测中的应用	115
第 8 章	线性神经网络算法分析与实现	120
8.1	线性神经网络工具箱函数	120
8.1.1	创建函数	120
8.1.2	学习函数	122
8.1.3	性能函数	124
8.2	线性神经网络模型及结构	125
8.3	线性神经网络的学习算法与训练	126
8.3.1	线性神经网络的学习算法	126
8.3.2	线性神经网络的训练	128
8.4	线性神经网络的滤波器	130
第 9 章	感知器网络算法分析与实现	133
9.1	单层感知器	133
9.1.1	单层感知器模型	133
9.1.2	单层感知器功能	134
9.1.3	单层感知器结构	136
9.1.4	单层感知器学习算法	137
9.1.5	单层感知器训练	138
9.1.6	单层感知器局限性	139
9.1.7	单层感知器的 MATLAB 实现	140
9.2	多层感知器	147
9.2.1	多层感知器模型	147
9.2.2	多层感知器设计方法	147
9.2.3	多层感知器的 MATLAB 实现	148



第 10 章 神经网络工具箱函数分析与应用	153
10.1 神经网络仿真函数	153
10.2 神经网络训练函数	155
10.2.1 train	156
10.2.2 trainb 函数	156
10.3 神经网络学习函数	158
10.4 神经网络初始函数	161
10.5 神经网络输入函数	163
10.6 神经网络的传递函数	165
10.7 神经网络求点积函数	168
第 11 章 BM 网络与 BSB 网络算法分析与实现	169
11.1 Boltzmann 神经网络	169
11.1.1 BM 网络的基本结构	169
11.1.2 BM 模型的学习	169
11.1.3 BM 网络的实现	172
11.2 BSB 神经网络	174
第 12 章 感知器网络工具箱函数及其应用	177
12.1 创建函数	177
12.2 显示函数	180
12.3 性能函数	181
第 13 章 RBF 神经网络算法分析与应用	186
13.1 RBF 神经网络模型	186
13.2 RBF 神经网络的数学基础	188
13.2.1 内插问题	188
13.2.2 正则化网络	189
13.3 RBF 神经网络的学习算法	190
13.3.1 自组织选取中心法	190
13.3.2 梯度训练方法	191
13.3.3 正交最小二乘 (OLS) 学习算法	192
13.4 其他 RBF 神经网络	193
13.4.1 广义回归神经网络	193
13.4.2 泛化回归神经网络	194
13.4.3 概率神经网络	195
13.5 RBF 神经网络 MATLAB 函数	196
13.5.1 创建函数	196
13.5.2 权函数	199
13.5.3 输入函数	200
13.5.4 传递函数	201

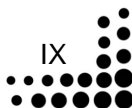




13.5.5	mse 函数	201
13.5.6	变换函数	202
第 14 章	Simulink 神经网络应用	204
14.1	Simulink 神经网络仿真模型库	204
14.2	Simulink 神经网络应用	208
第 15 章	ART 网络与 CP 网络算法分析与应用	213
15.1	ART-1 型网络	213
15.1.1	ART-1 型网络结构	213
15.1.2	ART-1 网络学习过程	215
15.1.3	ART-1 网络的应用	216
15.2	ART-2 型网络	218
15.2.1	网络结构与运行原理	219
15.2.2	网络的数学模型与学习算法	220
15.2.3	ART-2 型网络在系统辨识中的应用	222
15.3	CP 神经网络概述	223
15.3.1	CP 网络学习	224
15.3.2	CP 网络应用	225
第 16 章	Hopfield 网络算法分析与实现	231
16.1	Hopfield 神经网络	231
16.1.1	离散型 Hopfield 网络	231
16.1.2	DHNN 的动力学稳定性	234
16.1.3	网络权值的学习	236
16.1.4	联想记忆功能	239
16.2	连续型 Hopfield 网络	240
16.3	Hopfield 神经网络的应用	242
16.3.1	Hopfield 神经网络函数	242
16.3.2	Hopfield 神经网络的应用	245
第 17 章	LVQ 网络算法分析与应用	259
17.1	LVQ 神经网络的结构	259
17.2	LVQ 神经网络的学习算法	260
17.2.1	LVQ1 算法	260
17.2.2	LVQ2 算法	260
17.3	LVQ 神经网络的特点	261
17.4	LVQ 神经网络的 MATLAB 函数	262
17.5	LVQ 神经网络的应用	264
第 18 章	自组织网络算法分析与实现	269
18.1	竞争学习的概念	270
18.2	竞争学习规则	271

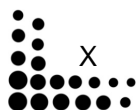


18.3	竞争学习原理	272
18.4	竞争神经网络 MATLAB 实现	275
18.5	竞争型神经网络存在的问题	279
第 19 章	Elman 网络算法分析与应用	280
19.1	Elman 神经网络结构	280
19.2	Elman 神经网络权值修正的学习算法	281
19.3	Elman 网络稳定性推导	282
19.4	对角递归网络稳定时学习速率的确定	283
19.5	Elman 神经网络在数据预测中的应用	284
第 20 章	BP 网络工具箱函数及其应用	288
20.1	创建函数	289
20.2	传递函数	291
20.3	学习函数	293
20.4	训练函数	294
20.5	性能函数	297
20.6	显示函数	298
第 21 章	神经网络在实际案例中的应用	300
21.1	农作物虫情预测	300
21.1.1	虫情预测原理	300
21.1.2	网络实现	301
21.2	人脸识别	304
21.2.1	模型建立	305
21.2.2	网络实现	306
第 22 章	神经网络工具箱函数分析与应用	310
22.1	神经网络构建函数的分析与应用	310
22.2	神经网络应用函数的分析与应用	324
第 23 章	线性神经网络算法分析与设计	330
23.1	线性神经网络结构	330
23.2	线性神经网络设计	331
23.3	自适应滤波线性神经网络	333
23.4	线性神经网络的局限性	335
23.5	线性神经网络的 MATLAB 应用举例	336
第 24 章	神经网络工具箱函数及实例分析	342
24.1	传递函数及其导函数	342
24.1.1	传递函数	342
24.1.2	传递函数的导函数	349
24.2	距离函数	354
24.3	权值函数及其导函数	356





24.3.1	权值函数	357
24.3.2	权值函数的导函数	358
24.4	结构函数	359
24.5	分析函数	361
24.6	转换函数	362
24.7	绘图函数	368
24.8	数据预处理和后处理函数	375
第 25 章	神经网络的工程应用	383
25.1	线性神经网络在线性预测中的应用	383
25.2	神经模糊控制在洗衣机中的应用	385
25.2.1	洗衣机的模糊控制	385
25.2.2	洗衣机的神经网络模糊控制器的设计	387
25.3	模糊神经网络在配送中心选址中的应用	391
25.4	Elman 神经网络在信号检测中的应用	394
25.5	神经网络在噪声抵消系统中的应用	397
25.5.1	自适应噪声抵消原理	397
25.5.2	噪声抵消系统的 MATLAB 仿真	399
第 26 章	神经网络算法分析与工具箱应用	402
26.1	网络对象属性	404
26.1.1	结构属性	404
26.1.2	子对象结构属性	408
26.1.3	函数属性	411
26.1.4	权值和阈值	413
26.1.5	参数属性	415
26.1.6	其他属性	415
26.2	子对象属性	416
26.2.1	输入向量	416
26.2.2	网络层	417
26.2.3	输出向量	422
26.2.4	阈值向量	422
26.2.5	输入权值向量	424
26.2.6	目标向量	427
26.2.7	网络层权值向量	428
第 27 章	自定义函数及其应用	432
27.1	初始化函数	432
27.2	学习函数	435
27.3	仿真函数	440
27.3.1	传递函数	440



27.3.2 传递函数导数函数..... 443

27.3.3 网络输入函数..... 444

27.3.4 网络输入导函数..... 446

27.3.5 权值函数..... 448

27.3.6 权值导数函数..... 450

27.4 自组织函数 452

27.4.1 拓扑函数..... 452

27.4.2 距离函数..... 454

参考文献..... 456

第 1 章 RBF 神经网络的实际应用

下面通过几个实例来演示 RBF 神经网络的应用。

1.1 用于曲线拟合的 RBF 神经网络

【例 1-1】 使用 NEWRB 的函数对接近的一组数据点创建径向基网络，完成 $y=f(x)$ 的曲线拟合。

```
>> clear all;  
%定义 21 个输入 P 和相关目标向量 T  
X = -1:1:1;  
T = [-.9602    -.5770   -.0729    .3771    .6405    .6600    .4609 ...  
      .1336    -.2013   -.4344   -.5000   -.3930   -.1647    .0988 ...  
      .3072    .3960    .3449    .1816   -.0312   -.2189   -.3201];  
plot(X,T,'+'); %效果如图 1-1 所示  
title('训练向量');  
xlabel('输入向量 P');  
ylabel('目标向量 T');
```

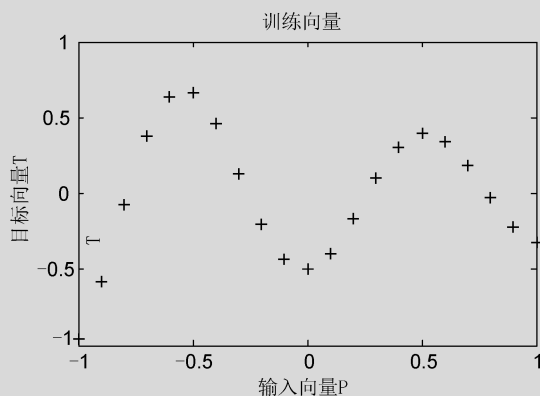


图 1-1 训练数据散点图

使用径向传递函数 radbas 计算隐藏层的输出，代码为：

```
>> x = -3:1:3;  
a = radbas(x);  
plot(x,a) %效果如图 1-2 所示
```



```
title('径向基传递函数');  
xlabel('输入向量 p');  
ylabel('输出向量 a');
```

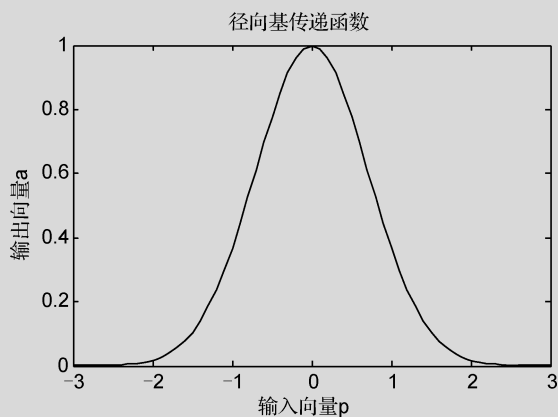


图 1-2 径向基传递函数

定义径向基网络权值与阈值不同的宽度，比较各隐藏层输出传递函数曲线，三个径向基函数用曲线用“蓝色”表示，其加权和用“洋红色”表示。代码为：

```
>> a2 = radbas(x-1.5);  
a3 = radbas(x+2);  
a4 = a + a2*1 + a3*0.5; %加权和  
plot(x,a,'b-',x,a2,'b--',x,a3,'b--',x,a4,'m-') %效果如图 1-3 所示  
title('径向基传递函数的加权和');  
xlabel('输入向量 p');  
ylabel('输出向量 a');
```

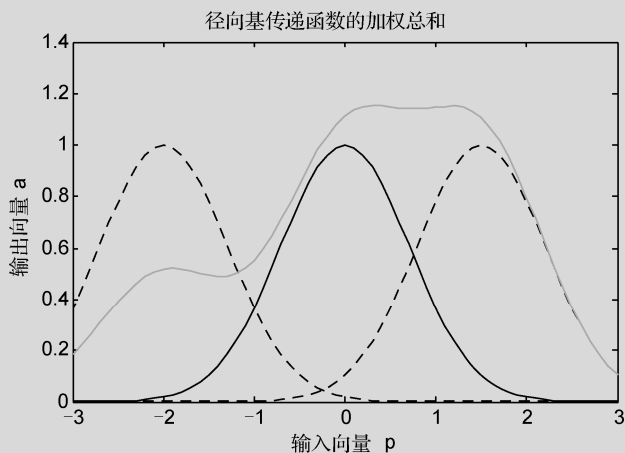


图 1-3 径向函数与加权和效果图

利用 NEWRB 快速创建一个接近 P 和 T 的定义除了训练集和目标函数的径向基网络，代码为：



```
>> eg = 0.02;           %总和平方误差目标
sc = 1;                 %扩展速度，默认值为 1
```

仿真过程如下，误差如图 1-4 所示。

```
net = newrb(X,T,eg,sc);
NEWRB, neurons = 0, MSE = 0.176192
NEWRB, neurons = 2, MSE = 0.160368
NEWRB, neurons = 3, MSE = 0.128338
NEWRB, neurons = 4, MSE = 0.0275185
NEWRB, neurons = 5, MSE = 0.0264878
NEWRB, neurons = 6, MSE = 0.00046188
```

绘制函数拟合曲线，代码如下：

```
>> plot(X,T,'+');
xlabel('输入');
X = -1:0.01:1;
Y = net(X);
hold on;
plot(X,Y);
hold off;
legend(['目标','输出'])
```

得到拟合曲线如图 1-5 所示。其中实线为得到的拟合曲线，“+”为训练样本。

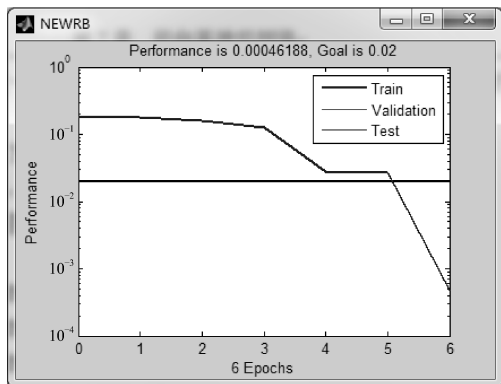


图 1-4 误差过程图

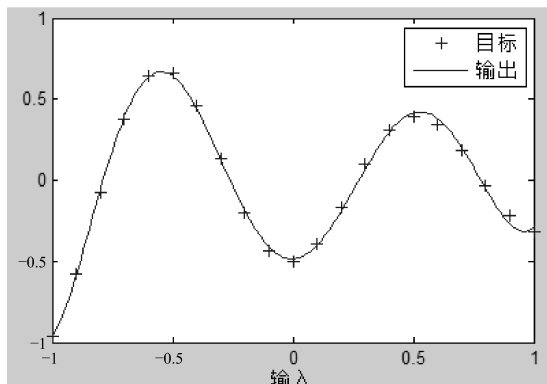


图 1-5 曲线拟合效果

【例 1-2】 下面给出基于聚类、梯度法和最小二乘法的 RBF 神经网络的设计算法，实现函数拟合。

(1) 首先是基于聚类的 RBF 神经网络的设计算法，代码为：

```
>> clear all;
SamNum = 100;           %总样本数
TestSamNum = 101;       %测试样本数
InDim = 1;              %样本输入维数
```



```
ClusterNum = 10; %隐节点数，即聚类样本数
Overlap = 1.0; %隐节点重叠系数
%根据目标函数获得样本输入输出
rand('state',sum(100*clock))
NoiseVar = 0.1;
Noise = NoiseVar*randn(1,SamNum);
SamIn = 8*rand(1,SamNum)-4;
SamOutNoNoise = 1.1*(1-SamIn+2*SamIn.^2).*exp(-SamIn.^2/2);
SamOut = SamOutNoNoise + Noise;
TestSamIn = -4:0.08:4;
TestSamOut = 1.1*(1-TestSamIn+2*TestSamIn.^2).*exp(-TestSamIn.^2/2);
figure
hold on
grid
plot(SamIn,SamOut,'k+')
plot(TestSamIn,TestSamOut,'k--')
xlabel('输入 x');
ylabel('输出 y');
Centers = SamIn(:,1:ClusterNum);
NumberInClusters = zeros(ClusterNum,1); %各类中的样本数，初始化为零
IndexInClusters = zeros(ClusterNum,SamNum); %各类所含样本的索引号
while 1,
    NumberInClusters = zeros(ClusterNum,1); %各类中的样本数，初始化为零
    IndexInClusters = zeros(ClusterNum,SamNum); %各类所含样本的索引号
    %按最小距离原则对所有样本进行分类
    for i = 1:SamNum
        AllDistance = dist(Centers',SamIn(:,i));
        [MinDist,Pos] = min(AllDistance);
        NumberInClusters(Pos) = NumberInClusters(Pos) + 1;
        IndexInClusters(Pos,NumberInClusters(Pos)) = i;
    end
    %保存旧的聚类中心
    OldCenters = Centers;
    for i = 1:ClusterNum
        Index = IndexInClusters(i,1:NumberInClusters(i));
        Centers(:,i) = mean(SamIn(:,Index))';
    end
    %判断新旧聚类中心是否一致，是则结束聚类
    EqualNum = sum(sum(Centers==OldCenters));
    if EqualNum == InDim*ClusterNum,
        break,
    end
```



```

end
%计算各隐节点的扩展常数（宽度）
AllDistances = dist(Centers',Centers);
Maximum = max(max(AllDistances));
for i = 1:ClusterNum
AllDistances(i,i) = Maximum+1;
end
Spreads = Overlap*min(AllDistances);
%计算各隐节点的输出权值
Distance = dist(Centers',SamIn);
SpreadsMat = repmat(Spreads,1,SamNum);
HiddenUnitOut = radbas(Distance./SpreadsMat);
HiddenUnitOutEx = [HiddenUnitOut' ones(SamNum,1)]';
W2Ex = SamOut*pinv(HiddenUnitOutEx);
W2 = W2Ex(:,1:ClusterNum);
B2 = W2Ex(:,ClusterNum+1);
%测试
TestDistance = dist(Centers',TestSamIn);
TestSpreadsMat = repmat(Spreads,1,TestSamNum);
TestHiddenUnitOut = radbas(TestDistance./TestSpreadsMat);
TestNNOOut = W2*TestHiddenUnitOut+B2;
plot(TestSamIn,TestNNOOut,'k-')
W2
B2

```

%计算隐节点数据中心间的距离（矩阵）

%找出其中最大的一个距离

%将对角线上的 0 替换为较大的值

%以隐节点间的最小距离作为扩展常数

%计算各样本输入离各数据中心的距离

%计算隐节点输出阵

%考虑偏移

%求广义输出权值

%输出权值

%偏移

运行程序，输出如下，效果如图 1-6 所示。

```

W2 =
-1.5956 -1.2670 0.4374 -0.2994 -0.0931 0.3085 -0.8392 -0.7220 -0.3804 -2.0981
B2 =
2.0542

```

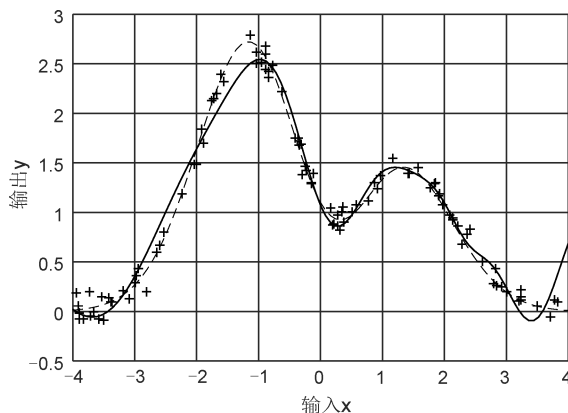


图 1-6 基于聚类的拟合效果图



其中，图中的“+”表示的是原始样本数据，实线为拟合的曲线。

(2) 基于梯度法的 RBF 神经网络设计算法，代码如下：

```
>> clear all;
SamNum = 100; %训练样本数
TargetSamNum = 101; %测试样本数
InDim = 1; %样本输入维数
UnitNum = 10; %隐节点数
MaxEpoch = 5000; %最大训练次数
E0 = 0.9; %目标误差
%根据目标函数获得样本输入输出
rand('state',sum(100*clock))
NoiseVar = 0.1;
Noise = NoiseVar*randn(1,SamNum);
SamIn = 8*rand(1,SamNum)-4;
SamOutNoNoise = 1.1*(1-SamIn+2*SamIn.^2).*exp(-SamIn.^2/2);
SamOut = SamOutNoNoise + Noise;
TargetIn = -4:0.08:4;
TargetOut = 1.1*(1-TargetIn+2*TargetIn.^2).*exp(-TargetIn.^2/2);
figure
hold on
grid
plot(SamIn,SamOut,'k+')
plot(TargetIn,TargetOut,'k--')
xlabel('输入 x');
ylabel('输出 y');
Center = 8*rand(InDim,UnitNum)-4;
SP = 0.2*rand(1,UnitNum)+0.1;
W = 0.2*rand(1,UnitNum)-0.1;
lrCent = 0.001; %隐节点数据中心学习系数
lrSP = 0.001; %隐节点扩展常数学习系数
lrW = 0.001; %隐节点输出权值学习系数
ErrHistory = []; %用于记录每次参数调整后的训练误差
for epoch = 1:MaxEpoch
    AllDist = dist(Center',SamIn);
    SPMat = repmat(SP,1,SamNum);
    UnitOut = radbas(AllDist./SPMat);
    NetOut = W*UnitOut;
    Error = SamOut-NetOut;
    %停止学习判断
    SSE = sumsqr(Error);
    %记录每次权值调整后的训练误差
    ErrHistory = [ErrHistory SSE];
```



```

if SSE<E0, break, end
for i = 1:UnitNum
    CentGrad = (SamIn-repmat(Center(:,i),1,SamNum))...
        *(Error.*UnitOut(i,:)*W(i)/(SP(i)^2));
    SPGrad = AllDist(i,:).^2*(Error.*UnitOut(i,:)*W(i)/(SP(i)^3));
    WGrad = Error*UnitOut(i,:);
    Center(:,i) = Center(:,i) + lrCent*CentGrad;
    SP(i) = SP(i) + lrSP*SPGrad;
    W(i) = W(i) + lrW*WGrad;
end
end
%测试
TestDistance = dist(Center',TargetIn);
TestSpreadsMat = repmat(SP',1,TargetSamNum);
TestHiddenUnitOut = radbas(TestDistance./TestSpreadsMat);
TestNNOOut = W*TestHiddenUnitOut;
plot(TargetIn,TestNNOOut,'k-')
%绘制学习误差曲线
figure
hold on
grid
[xx,Num] = size(ErrHistory);
plot(1:Num,ErrHistory,'k-');

```

运行程序，先得到网络收敛过程图，如图 1-7 所示，拟合效果如图 1-8 所示。

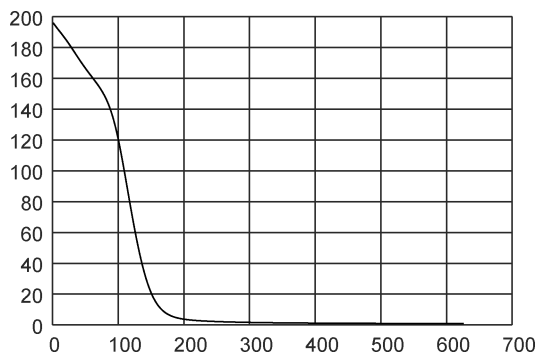


图 1-7 网络收敛过程图

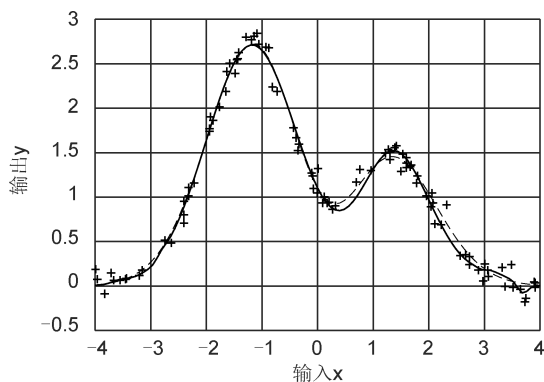


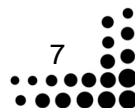
图 1-8 基于梯度法的曲线拟合效果

(3) 基于最小二乘法的 RBF 神经网络算法，实现代码为：

```

>> clear all;
SamNum = 100;           %训练样本数
TestSamNum = 101;       %测试样本数
SP = 0.6;               %隐节点扩展常数

```





```
ErrorLimit = 0.9; % 目标误差
% 根据目标函数获得样本输入输出
rand('state',sum(100*clock))
NoiseVar = 0.1;
Noise = NoiseVar*randn(1,SamNum);
SamIn = 8*rand(1,SamNum)-4;
SamOutNoNoise = 1.1*(1-SamIn+2*SamIn.^2).*exp(-SamIn.^2/2);
SamOut = SamOutNoNoise + Noise;
TestSamIn = -4:0.08:4;
TestSamOut = 1.1*(1-TestSamIn+2*TestSamIn.^2).*exp(-TestSamIn.^2/2);
figure
hold on
grid
plot(SamIn,SamOut,'k+')
plot(TestSamIn,TestSamOut,'k--')
xlabel('输入 x');
ylabel('输出 y');
[InDim,MaxUnitNum] = size(SamIn); % 样本输入维数和最大允许隐节点数
% 计算隐节点输出阵
Distance = dist(SamIn',SamIn);
HiddenUnitOut = radbas(Distance/SP);
PosSelected = [];
VectorsSelected = [];
HiddenUnitOutSelected = [];
ErrHistory = []; % 用于记录每次增加隐节点后的训练误差
VectorsSelectFrom = HiddenUnitOut;
dd = sum((SamOut.*SamOut)');
for k = 1: MaxUnitNum
    % 计算各隐节点输出向量与目标输出向量的夹角平方值
    PP = sum(VectorsSelectFrom.*VectorsSelectFrom)';
    Denominator = dd * PP';
    [xxx,SelectedNum] = size(PosSelected);
    if SelectedNum>0,
        [lin,xxx] = size(Denominator);
        Denominator(:,PosSelected) = ones(lin,1);
    end
    Angle = ((SamOut*VectorsSelectFrom).^2) ./ Denominator;
    % 选择具有最大投影的向量，得到相应的数据中心
    [value,pos] = max(Angle);
```




```

PosSelected = [PosSelected pos];
%计算 RBF 网训练误差
HiddenUnitOutSelected = [HiddenUnitOutSelected; HiddenUnitOut(pos,:)];
HiddenUnitOutEx = [HiddenUnitOutSelected; ones(1,SamNum)];
W2Ex = SamOut*pinv(HiddenUnitOutEx);    %用广义逆求广义输出权值
W2 = W2Ex(:,1:k);                      %得到输出权值
B2 = W2Ex(:,k+1);                      %得到偏移
NNOOut = W2*HiddenUnitOutSelected+B2;    %计算 RBF 网输出
SSE = sumsq(SamOut-NNOOut);
%记录每次增加隐节点后的训练误差
ErrHistory = [ErrHistory SSE];
if SSE < ErrorLimit,
    break,
end
%作 Gram-Schmidt 正交化
NewVector = VectorsSelectFrom(:,pos);
ProjectionLen = NewVector' * VectorsSelectFrom / (NewVector'*NewVector);
VectorsSelectFrom = VectorsSelectFrom - NewVector * ProjectionLen;
end
UnitCenters = SamIn(PosSelected);
%测试
TestDistance = dist(UnitCenters',TestSamIn);
TestHiddenUnitOut = radbas(TestDistance/SP);
TestNNOOut = W2*TestHiddenUnitOut+B2;
plot(TestSamIn,TestNNOOut,'k-')
k
UnitCenters
W2
B2

```

运行程序，输出如下，拟合效果如图 1-9 所示。

```

k =
    9
UnitCenters =
   -1.1438   1.3982  -0.0193  -2.0332   2.3536  -0.5274   0.8565  -1.3504   1.7926
W2 =
   1.2668   0.7839   0.4824   1.0814   0.4007   1.0261   0.6510   1.0261   0.4116
B2 =
   0.0664

```

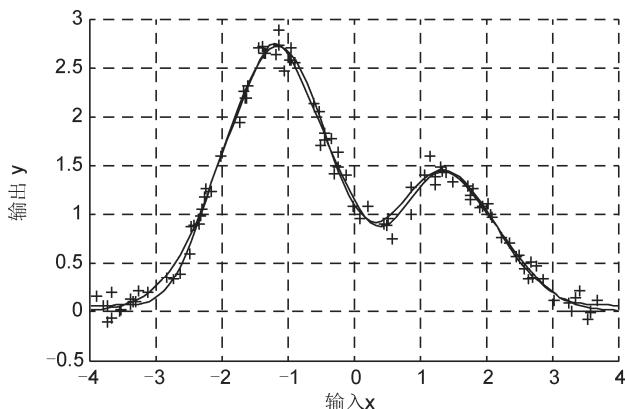


图 1-9 基于最小二乘法的拟合效果

1.2 径向基网络实现非线性函数回归

【例 1-3】 使用不同径向基网络实现非线性函数的回归分析。

(1) 使用 exact 径向基函数实现非线性函数回归分析，实现代码为：

```
>> clear all;
%产生输入输出数据
%设置步长
interval=0.01;
%产生 x1 x2
x1=-1.5:interval:1.5;
x2=-1.5:interval:1.5;
%按照函数先求得相应的函数值，作为网络的输出
F=20+x1.^2-10*cos(2*pi*x1)+x2.^2-10*cos(2*pi*x2);
%网络建立和训练
%网络建立输入为[x1;x2]，输出为 F。spread 使用默认值 1
net=newrbe([x1;x2],F);
%将原数据回代，测试网络效果
ty=sim(net,[x1;x2]);
%使用图像来查看网络对非线性函数的拟合效果
plot3(x1,x2,F,'rd');
hold on;
plot3(x1,x2,ty,'b-.');
view(113,36)
title('可视化的方法观察严格的 RBF 神经网络的拟合效果')
xlabel('x1');
ylabel('x2');
zlabel('F');
```



```
grid on;
```

运行程序，效果如图 1-10 所示。

可视化的方法观察严格的RBF神经网络的拟合效果

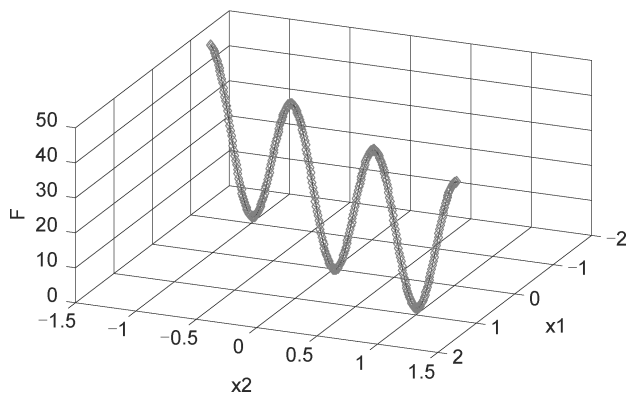
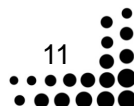


图 1-10 exact 径向基网络拟合效果

(2) 用 approximate RBF 神经网络对同一函数进行拟合，代码为：

```
>> clear all;
%产生训练样本（训练输入，训练输出）
%ld 为样本例数
ld=400;
%产生 2*ld 的矩阵
x=rand(2,ld);
%将 x 转换到[-1.5 1.5]之间
x=(x-0.5)*1.5*2;
%x 的第一行为 x1，第二行为 x2
x1=x(1,:);
x2=x(2,:);
%计算网络输出 F 值
F=20*x1.^2-10*cos(2*pi*x1)+x2.^2-10*cos(2*pi*x2);
%建立 RBF 神经网络
%采用 approximate RBF 神经网络，spread 为默认值
net=newrb(x,F);
%建立测试样本
interval=0.1;
[i,j]=meshgrid(-1.5:interval:1.5);
row=size(i);
tx1=i(:);
tx1=tx1';
tx2=j(:);
tx2=tx2';
tx=[tx1;tx2];
```





```
%使用建立的 RBF 神经网络进行模拟，得出网络输出
ty=sim(net,tx);
%使用图像，给出三维图
interval=0.1;
[x1,x2]=meshgrid(-1.5:interval:1.5);
F=20*x1.^2-10*cos(2*pi*x1)+x2.^2-10*cos(2*pi*x2);
subplot(131);mesh(x1,x2,F);
title('真正的函数图像');
%网络得出的函数图像
v=reshape(ty,row);
subplot(1,3,2);mesh(i,j,v);
zlim([0 60]);
title('RBF 神经网络图像');
%误差图像
subplot(1,3,3);mesh(x1,x2,'F-v');
zlim([0 60]);
title('误差图像');
```

运行程序，仿真过程如下所示，效果如图 1-11 所示。

```
set(gcf,'position',[300 250 900 400]);
NEWRB, neurons = 0, MSE = 278.927
NEWRB, neurons = 2, MSE = 233.328
NEWRB, neurons = 3, MSE = 139.497
NEWRB, neurons = 4, MSE = 139.175
NEWRB, neurons = 5, MSE = 129.179
NEWRB, neurons = 6, MSE = 125.342
NEWRB, neurons = 7, MSE = 113.113
NEWRB, neurons = 8, MSE = 111.769
.....
NEWRB, neurons = 398, MSE = 3.80993e-09
NEWRB, neurons = 399, MSE = 9.50774e-06
NEWRB, neurons = 400, MSE = 5.21143e-09
```

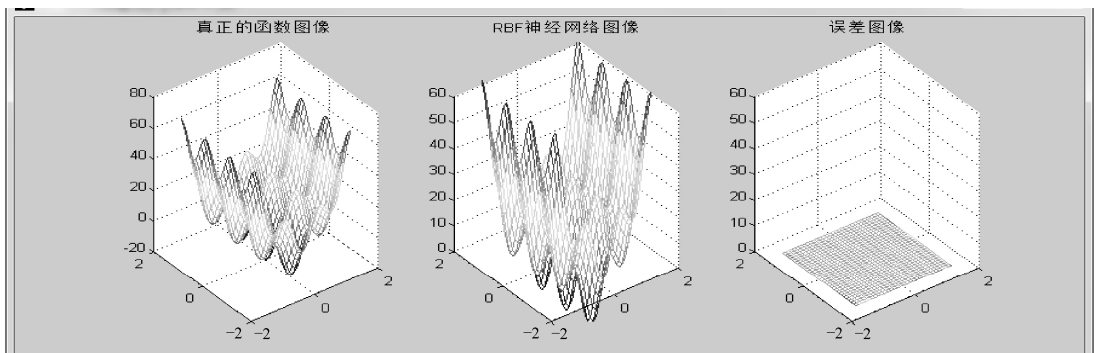


图 1-11 approximate RBF 神经网络拟合效果比较图与误差图



由图 1-10 及图 1-11 可知,神经网络的训练结果能较好地逼近该非线性函数 F ,由误差图上看,神经网络的预测效果在数据边缘处的误差较大;在其他数值处的拟合效果很好。网络的输出和函数值之间的差值在隐藏层神经元的个数为 100 时已经接近于 0,说明网络输出能非常好地逼近函数。

1.3 CRNN 网络应用

【例 1-4】 泛化神经网络通常用来进行函数逼近,本例就是演示如何应用 CRNN 进行函数逼近。

假设选取函数的 8 个数据点,输入向量为 P ,输出的期望值为 T 。目的就是应用 `newgrnn` 函数设计 CRNN 网络对该样本实现函数逼近。

```
>>clear all;
P=[1 2 3 4 5 6 7 8];
T=[0 1 2 3 2 1 2 1];
%绘制出输入/输出样本点,如图 1-12 所示。
plot(P,T,'.', 'markersize',30)
axis([0 9 -1 4])
title('待逼近函数')
xlabel('P')
ylabel('T')
```

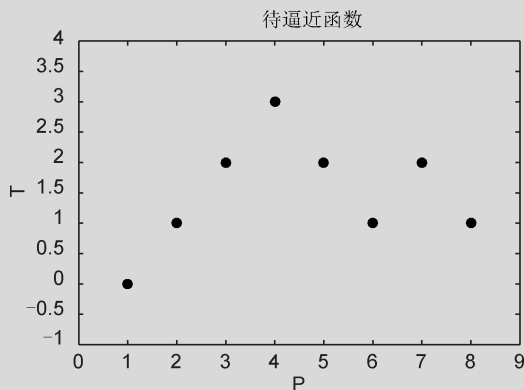


图 1-12 待逼近函数

应用 `newgrnn` 函数构建一个广义回归神经网络,对于离散数据点的情况,散布常数 `spread` 的选取比输入向量之间的距离 1 稍小一些,散布常数越小,数据拟合越好,但是曲线光滑度要差些。

```
spread=0.7;
net=newgrnn(P,T,spread);
```

网络构建后,对应输入向量 P 应用 `sim` 函数进行仿真,并将网络输出和期望值绘制在一



张图上，如图 1-13 所示。

```
A=sim(net,P);  
hold on  
outputline=plot(P,A,'o','markersize',10,'color',[1 0 0]);  
title('检测网络')  
xlabel('P')  
ylabel('T 和 A')
```

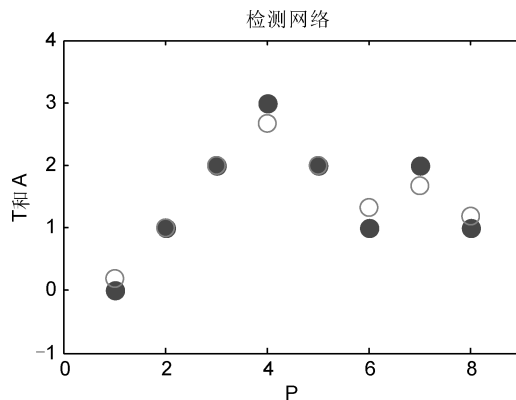


图 1-13 网络输出和期望值

图 1-13 中的圆点代表期望值，圆圈代表网络输出值。从图中可以看出，网络基本上做了函数逼近。下面再通过新的输入向量 P 进行测试、仿真并绘制出测试结果，如图 1-14 所示，“+”表示新输入值。

```
p=3.5;  
a=sim(net,p);  
plot(p,a,'+','markersize',10,'color',[1 0 0]);  
title('新输入值')  
xlabel('P 和 p')  
ylabel('T 和 a')
```

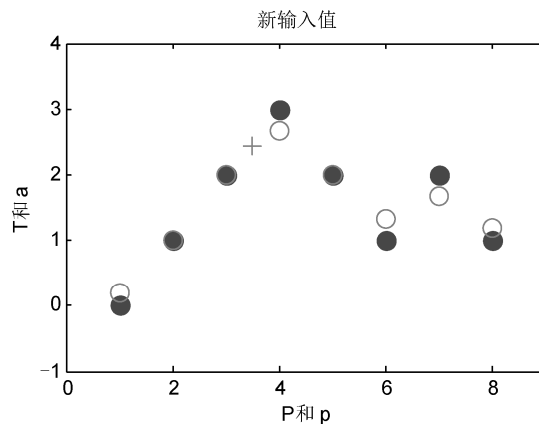


图 1-14 对新的输入值仿真



增加采样点，绘制拟合的函数曲线，如图 1-15 所示。

```
P2=0:1:9;
A2=sim(net,P2);
plot(P2,A2,'linewidth',4,'color',[1 0 0])
title('逼近函数')
xlabel('P 和 P2')
ylabel('T 和 A2')
```

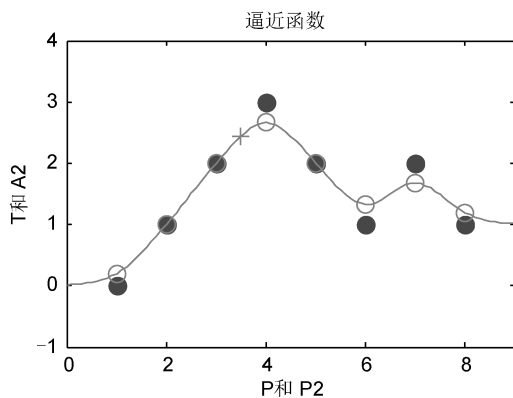


图 1-15 逼近函数

1.4 PNN 网络应用

【例 1-5】以 PNN 完成如图 1-16 所示的两类模式的分类。

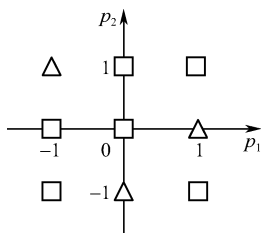


图 1-16 待分类模式

将正方形规定为第 1 类模式，三角形规定为第 2 类模式。以 (p_1, p_2) 代表各模式样本的位置，形成相应的输入向量。

其设计 PNN 的 MATLAB 程序代码如下：

```
>> clear all;
%定义输入向量和目标向量
p=[0 0 1 1 1 1 -1 -1 -1;0 1 -1 0 1 -1 0 1 -1];
t=[1 1 2 2 1 1 1 2 1];
t=ind2vec(t);
```



```
% 设计 PNN
t1=clock;                                % 计时开始
net=newpnn(p,t,0.7);
datat=etime(clock,t1)                   % 计算设计网络所用的时间
% 存储训练好的神经网络
```

运行程序，输出网络设计时间为：

```
datat =
    0.4690
```

实现 PNN 的 MATLAB 仿真程序如下：

```
>> clear all;
% 定义待测试样本输入向量
p=[0 0 0 1 1 1 -1 -1 -1;0 1 -1 0 1 -1 0 1 -1];
% 加载训练好的神经网络
load li5_10 net;
% 神经网络仿真,对待测试样本进行分类
y=sim(net,p);
yc=vec2ind(y)
```

运行程序，输出如下：

```
yc =
     1     1     2     2     1     1     1     2     1
```

结果很好地完成了分类。读者不妨采用不同的径向基扩展常数 `spread` 设计该网络，看看分类结果有什么不同。

【例 1-6】 概率神经网络主要应用于模式识别、分类等问题中，本例演示如何应用 PNN 网络进行变量分类。

三组二维输入向量 **P**，以及其相对应的三个类别（期望类别）**Tc**，目的是构建一个 PNN 网络对输入向量进行正确分类。

```
>> clear all;
P=[1 2;2 2;1 1];
Tc=[1 2 3];
```

绘制输入向量及其相对应的类别，如图 1-17 所示。

```
plot(X(1,:),X(2,:),',' markersize',30)
for i=1:3, text(X(1,i)+0.1,X(2,i),sprintf('class %g',Tc(i))), end
axis([0 3 0 3])
title('三向量及其类别')
xlabel('X(1,:)')
ylabel('X(2,:)')
```

首先将期望类别指针 **Tc** 转换为向量 **T**，然后应用 `newpnn` 函数设计一个概率神经网络。因为输入向量之间的距离为 1，所以设定散布常数为 1。

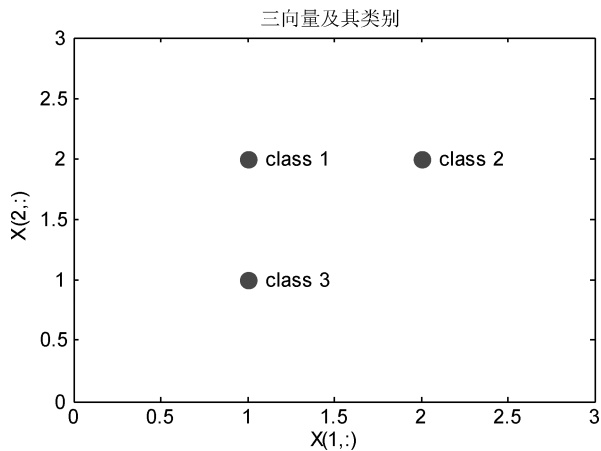


图 1-17 输入向量及类别

```
T=ind2vec(Tc);
spread=1;
net=newpnn(P,T,spread);
```

应用 `sim` 函数对输入向量进行仿真，并将网络输出向量转换为指针。

```
A=sim(net,P);
Ac=vec2ind(A);
```

然后绘制输入向量及相应的网络输出，即通过网络仿真得到向量分类结果，如图 1-18 所示。

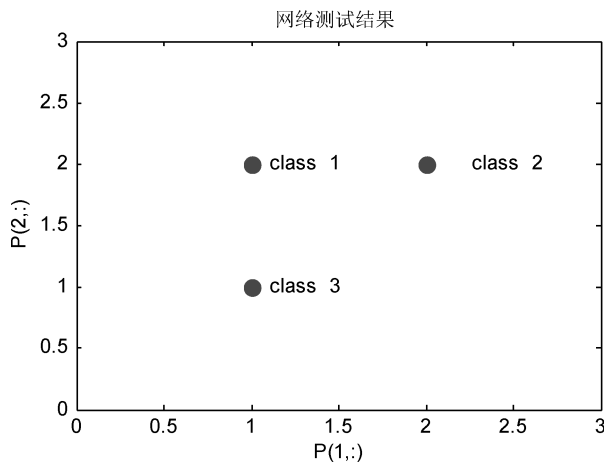


图 1-18 网络测试结果 1

```
plot(P(1,:),P(2,:),'','markersize',30)
axis([0 3 0 3])
for i=1:3
    text(P(1,i)+0.1,P(2,i),sprintf('class %g',Ac(i)));
```



```

end
title('网络测试结果')
xlabel('P(1,:)')
ylabel('P(2,:)')

```

从图中可以看出，与期望类别完全一致，可见网络设计是成功的。下面再对新的输入向量 p 进行测试，仿真并绘制出测试结果，如图 1-19 所示。

```

p=[2;1.5];
a=sim(net,p);
ac=vec2ind(a);
hold on
plot(p(1),p(2),'*','markersize',10,'color',[1 0 0])
text(p(1)+0.1,p(2),sprintf('class %g',ac))
hold off
title('对新向量分类')
xlabel('p(1,:) 与 p(1)')
ylabel('p(2,:) 与 p(2)')

```

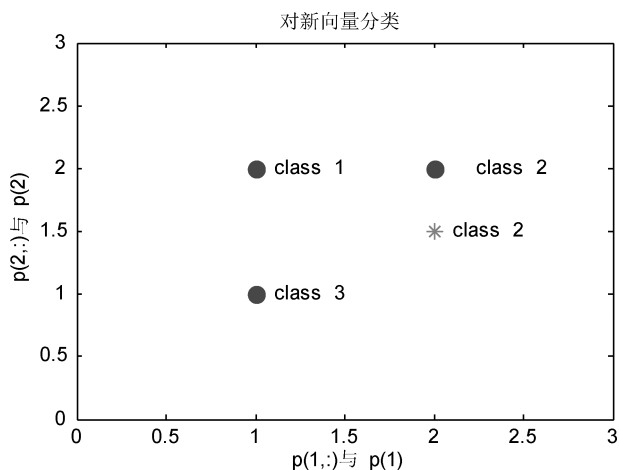


图 1-19 网络测试结果 2

下面通过立体图显示三个类别的情况，以及新的输入向量对应的类别，如图 1-20 所示。

```

p1=0:.05:3;
p2=p1;
[p1,p2]=meshgrid(p1,p2);
pp=[p1(:),p2(:)]';
aa=sim(net,pp);
aa=full(aa);
m=mesh(p1,p2,reshape(aa(1,:),length(p1),length(p2)));
set(m,'facecolor',[0 0.5 1],'linestyle','none');
hold on

```



```

m=mesh(p1,p2,reshape(aa(2,:),length(p1),length(p2)));
set(m,'facecolor',[0 0.1 0.5],'linestyle','none');
m=mesh(p1,p2,reshape(aa(3,:),length(p1),length(p2)));
set(m,'facecolor',[0.5 0 1],'linestyle','none');
plot3(p(1,:),p(2,:),[1 1 1]+0.1,'.', 'markersize',30)
plot3(p(1),p(2),1.1,'*', 'markersize',10,'color',[1 0 0])
hold off
view(2)
title('向量分类立体图')
xlabel('p(1,:) 与 p(1)')
ylabel('p(2,:) 与 p(2)')

```

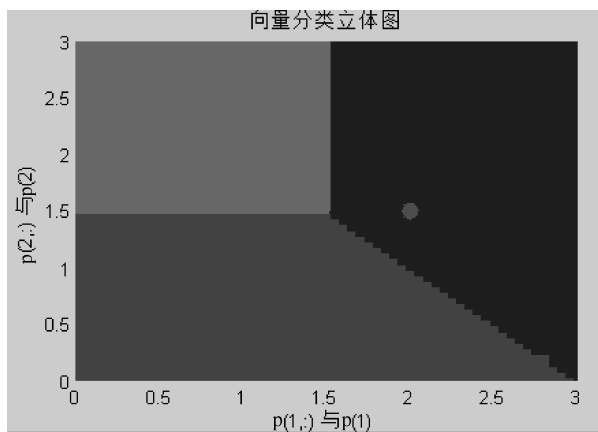


图 1-20 向量分类立体图

图 1-20 形象地显示了应用 PNN 网络进行向量分类的具体实现。

1.5 RBF 神经网络的优缺点

1. 优点

神经网络有很强的非线性拟合能力，可映射任意复杂的非线性关系，而且学习规则简单，便于计算机实现，具有很强的鲁棒性、记忆能力、非线性映射能力以及强大的自学习能力，因此有很大的应用市场。

具有局部逼近的优点：

RBF（径向基函数）神经网络是一种性能优良的前馈型神经网络，RBF 神经网络可以任意精度逼近任意非线性函数，且具有全局逼近能力，从根本上解决了 BP 神经网络的局部最优问题，而且拓扑结构紧凑，结构参数可实现分离学习，收敛速度快。RBF 神经网络和模糊逻辑能够实现很好的互补，提高神经网络的学习泛化能力。

RBF 神经网络的优点：



- (1) 具有唯一最佳逼近的特性，且无局部极小问题存在。
- (2) 具有较强的输入和输出映射功能，并且理论证明在前向网络中，RBF 神经网络是完成映射功能的最优网络。
- (3) 网络连接权值与输出呈线性关系。
- (4) 分类能力好。
- (5) 学习过程收敛速度快。

RB 神经除具有一般神经网络的优点，如多维非线性映射能力、泛化能力、并行信息处理能力等外，还具有很强的聚类分析能力、学习算法简单方便等优点。

径向基函数 (RBF) 神经网络是一种性能良好的前向网络，利用在多维空间中插值的传统技术，可以对几乎所有的系统进行辨识和建模，它不仅在理论上有着任意逼近性能和最佳逼近性能，而且在应用中具有很多优势。如 Sigmoid 函数作为激活函数的神经网络，算法速度大大高于一般的 BP 算法。RBF 神经网络同 BP 神经网络相比，不但在理论上是前向网络中最优的网络，而且学习方法也避免了局部最优的问题。

一个 RBF 神经网络，在隐层节点足够多的情况下，经过充分学习，可以用任意精度逼近任意非线性函数，而且具有最优泛函数逼近能力；另外，它具有较快的收敛速度、强大的抗噪和修复能力。

在理论上，RBF 神经网络和 BP 神经网络一样能以任意精度逼近任何非线性函数。但由于它们使用的激励函数不同，其逼近性能也不相同。Poggio 和 Girosi 已经证明，RBF 神经网络是连续函数的最佳逼近，而 BP 神经网络不是。BP 神经网络使用的 Sigmoid 函数具有全局特性，在输入值的很大范围内每个节点都对输出值产生影响，并且激励函数在输入值的很大范围内相互重叠，因而相互影响，因此 BP 神经网络训练过程很长。此外，由于 BP 算法的固有特性，BP 神经网络容易陷入局部极小的问题不可能从根本上避免，并且 BP 神经网络隐层节点数目的确定依赖于经验和试凑法，很难得到最优网络。采用局部激励函数的 RBF 神经网络在很大程度上克服了上述缺点，RBF 不仅有良好的泛化能力，而且对于每个输入值，只有很少几个节点具有非零激励值，因此只需改变很少部分节点及权值。学习速度可以比通常的 BP 算法提高上千倍，容易适应新数据，其隐层节点的数目也在训练过程中确定，并且其收敛性也较 BP 神经网络易于保证，因此可以得到最优解。

2. 缺点

RBF 神经网络具有以下缺点：

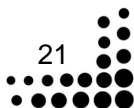
- (1) 最严重的问题是没能解释自己的推理过程和推理依据。
- (2) 不能向用户提出必要的询问，而且当数据不充分的时候，神经网络就无法进行工作。
- (3) 把一切问题的特征都变为数字，把一切推理都变为数值计算，其结果势必是丢失信息。
- (4) 理论和学习算法还有待于进一步完善和提高。

RBF 神经网络的非线性映射能力体现在隐层基函数上，而基函数的特性主要由基函数的中心确定，从数据点中任意选取中心构造出来的 RBF 神经网络的性能显然是不能令人满意的。



(5) RBF 神经网络用于非线性系统建模需要解决的关键问题是样本数据的选择。在实际工业过程中,系统的信息往往只能从系统运行的操作数据中分析得到。因此,如何从系统运行的操作数据中提取系统运行状况信息,以降低网络对训练样本的依赖,在实际应用中具有重要的价值。

隐层基函数的中心是在输入样本集中选取的,这在许多情况下难以反映系统真正的输入输出关系,并且初始中心点数太多;另外优选过程会出现数据病态现象。



第2章 SOM 网络算法分析与应用

1981 年，芬兰 Helsinki 大学的 T.Kohonen 教授提出一种自组织 SOM 网络，简称 SOM 网络，又称 Kohonen 网络。Kohonen 认为，一个神经网络接收外界输入模式时，将会分为不同的对应区域，各区域对输入模式具有不同的响应特征，而且这个过程是自动完成的。自组织特征映射正是根据这一看法提出来的，其特点与人脑的自组织特性相类似。

2.1 SOM 网络的生物学基础

生物学研究的事实表明，在人脑的感觉通道上，神经元的组织原理是有序排列。因此当人脑通过感官接收外界的特定时空信息时，大脑皮层的特定区域兴奋，而且类似的外界信息在对应区域是连续映像的。例如，生物视网膜中有许多特定的细胞对特定的图形比较敏感，当视网膜中有若干个接收单元同时受特定模式刺激时，就使大脑皮层中的特定神经元开始兴奋，输入模式接近，对应的兴奋神经元也相近。在听觉通道上，神经元在结构排列上与频率的关系十分密切，对于某个频率，特定的神经元具有最大的响应，位置邻近的神经元具有相近的频率特征，而远离的神经元具有的频率特征差别也较大。大脑皮层中神经元的这种响应特点不是先天安排的，而是通过后天的学习自组织形成的。

对于某一图形或某一频率的特定兴奋过程，神经元的有序排列以及对外界信息的连续映像是自组织特征映射网中竞争机制的生物学基础。当外界输入不同的样本时，网络中那个位置的神经元兴奋在训练开始时是随机的。但自组织训练后会在竞争层形成神经元的有序排列，功能相近的神经元非常靠近，功能不同的神经元离得较远。这一特点与人脑神经元的组织原理十分相似。

2.2 SOM 网络的拓扑结构

SOM 网络的拓扑结构如图 2-1 所示。

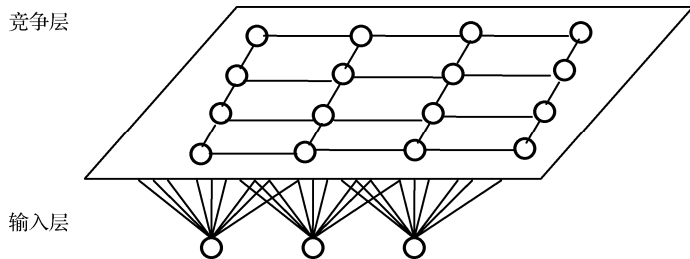


图 2-1 SOM 网络的拓扑结构



SOM 网络的一个典型特性是可以在一维或二维的处理单元阵列上, 形成输入信号的特征拓扑分布, 因此 SOM 网络具有抽取输入信号模式特征的能力。SOM 网络一般只包含一维阵列和二维阵列, 但也可以推广到多维处理单元阵列中。下面只讨论应用较多的二维阵列。SOM 网络模型由以下 4 部分组成。

- 处理单元阵列。用于接收事件输入, 并且形成对这些信号的“判别函数”。
- 比较选择机制。用于比较“判别函数”, 并选择一个具有最大函数输出值的处理单元。
- 局部互连作用。用于同时激励被选择的处理单元及其最邻近的处理单元。
- 自适应过程。用于修正被激励的处理单元的参数, 以增加其对应于特定输入“判别函数”的输出值。

假定网络输入为 $X \in R^n$, 输出神经元 i 与输入单元的连接权值为 $W_i \in R^n$, 则输出神经元 i 的输出 o_i 为:

$$o_i = W_i X \quad (2-1)$$

网络实际具有响应的输出单元 k , 该神经元的确定是通过“赢者通吃”的竞争机制得到的, 其输出为:

$$o_k = \max_i \{o_i\} \quad (2-2)$$

以上两式可修正为:

$$o_i = \sigma \left(\varphi_i + \sum_{t \in S_t} r_k o_t \right), \quad \varphi_i = \sum_{j=1}^m w_{ij} x_j, \quad o_k = \max_i \{o_i\} - \varepsilon$$

其中, w_{ij} 为输出神经元 i 和输入神经元 j 之间的连接权值。 x_j 为输入神经元 j 的输出。 $\sigma(t)$ 为非线性函数, 即

$$\sigma(t) = \begin{cases} 0 & t < 0 \\ \sigma(t) & 0 \leq t \leq A \\ A & t > A \end{cases} \quad (2-3)$$

ε 为一个很小的正数, r_k 为系数, 它与权值及横向连接有关。 S_i 为与处理单元 i 相关的处理单元集合, o_k 称为浮动阈值函数。

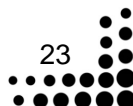
2.3 SOM 网络的权值调整

SOM 网络采用的算法称为 Kohonen 算法, 它是在“胜者为王”(Winner-Take-All, WTA) 学习规则基础上加以改进的, 主要区别是调整权向量与侧抑制的方式不同。

WTA: 侧抑制是“封杀”式的。只有获胜神经元可以调整其权值, 其他神经元都无权调整。

Kohonen 算法: 获胜的神经元对其邻近神经元的影响是由近及远, 由兴奋逐渐变为抑制。换句话说, 不仅获胜神经元要调整权值, 它周围的神经元也要不同程度地调整权向量。常见的调整方式有如下几种。

墨西哥草帽函数: 获胜节点有最大的权值调整量, 邻近的节点有稍小的调整量, 离获胜



节点距离越大, 权值调整量越小, 直到某一距离 d_0 时, 权值调整量为零; 当距离再远一些时, 权值调整量稍负, 更远又回到零, 如图 2-2 (a) 所示。

大礼帽函数: 它是墨西哥草帽函数的一种简化, 如图 2-2 (b) 所示。

厨师帽函数: 它是大礼帽函数的一种简化, 如图 2-2 (c) 所示。

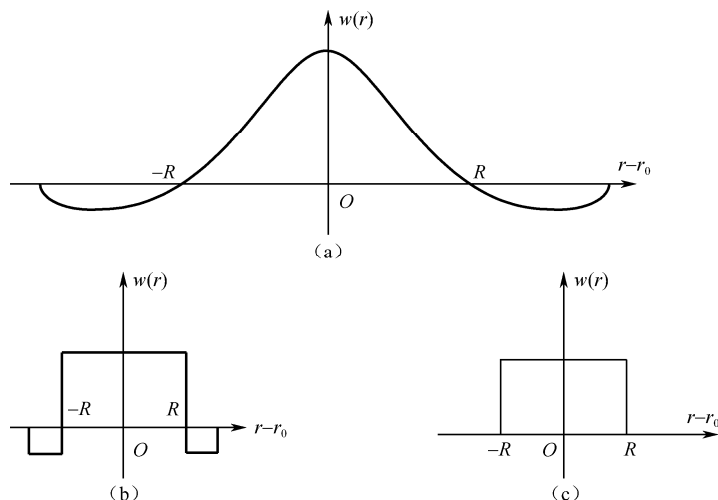


图 2-2 权值调整函数

以获胜神经元为中心设定一个邻域半径 R , 该半径固定的范围称为优胜邻域。在 SOM 网络学习方法中, 优胜邻域内的所有神经元均按其与其与获胜神经元的距离调整权值。优胜邻域开始定得较大, 但其大小随着训练次数的增加不断收缩, 最终收缩到半径为零。

Kohonen 算法能够自动找出输入数据之间的类似度, 将相似的输入在网络上就近配置, 因此是一种可以构成对输入数据有选择地给予反应的神经网络。Kohonen 算法的步骤如下所述。

(1) 网络初始化。

用随机数设定输入层和映射层之间权值的初始值。对 m 个输入神经元到输出神经元的连接权值赋予较小的权值。选取输出神经元 j 个“邻接神经元”的集合 S_j 。其中, $S_j(0)$ 表示时刻 $t=0$ 神经元 j 的“邻接神经元”的集合, $S_j(t)$ 表示时刻 t “邻接神经元”的集合。区域 $S_j(t)$ 随着时间的增加而不断缩小。

(2) 输入向量的输入。

把输入向量 $\mathbf{x} = (x_1, x_2, x_3, \dots, x_n)^T$ 输入给输入层。

(3) 计算映射层的权值向量和输入向量的距离 (欧式距离)。

在映射层, 计算各神经元的权值向量和输入向量的欧式距离。映射层的第 j 个神经元和输入向量的距离为:

$$d_j = \|\mathbf{X} - \mathbf{W}_j\| = \sqrt{\sum_{i=1}^m (x_i(t) - w_{ij}(t))^2} \quad (2-4)$$

式中, w_{ij} 为输入层的 i 神经元和映射层的 j 神经元之间的权值。通过计算, 得到一个具有最小距离的神经元, 将其作为获胜神经元, 记为 j^* , 即确定某个单元 k , 使得对于任意的 j ,



都有 $d_k = \min_j(d_j)$ ，并给出其邻接神经元集合。

(4) 定义优胜邻域 $S_{j^*}(t)$ 。

以 j^* 为中心确定 t 时刻的权值调整域，一般初始邻域 $S_{j^*}(0)$ 较大（为总节点的 50%～80%），训练过程中， $S_{j^*}(t)$ 随训练时间收缩，如图 2-3 所示。

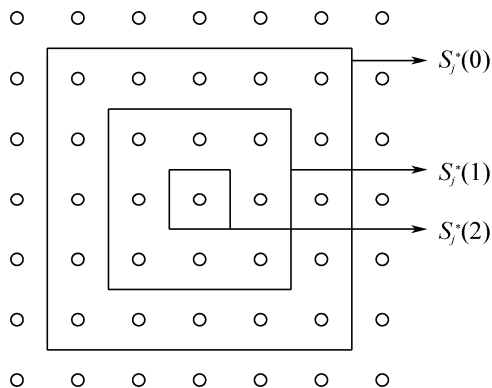


图 2-3 邻域 $S_j(t)$ 的收缩

(5) 权值的学习。

获胜神经元和位于其邻接神经元的权值按式 (2-5) 更新：

$$\Delta w_{ij} = \eta h(j, j^*)(x_i - w_{ij}) \quad (2-5)$$

式中， η 为一个大于 0 小于 1 的常数。

$h(j, j^*)$ 为领域函数，用式 (2-6) 表示：

$$h(j, j^*) = \exp\left(-\frac{|j - j^*|^2}{\sigma^2}\right) \quad (2-6)$$

式中的 σ^2 随着学习的进行而减小。因此， $h(j, j^*)$ 的范围学习初期很宽，随着学习的进行而变窄。也就是说，随着学习的进行从粗调整向微调变化。这样，领域函数 $h(j, j^*)$ 可以起到产生有效映射的作用。

(6) 计算输出 o_k 。

计算输出的表达式为：

$$o_k = f(\min_j \|X - W_j\|)$$

式中， $f(\cdot)$ 一般为 0~1 函数或者其他非线性函数。

(7) 如达到要求则算法结束，否则返回步骤 (2)，进入下一轮学习。

SOM 网络的结构和映射算法研究表明，大脑皮层的信息具有两个明显的特点：

其一，拓扑映射结构不是通过神经元的运动重新组织实现的，而是由各个神经元在不同兴奋状态下构成一个整体所形成的拓扑结构。

其二，这种拓扑映射结构的形成具有自组织的特点。SOM 网络中神经元的拓扑组织就是它最基本的特征。对于拓扑相关而形成的神经元子集，权重的更新是相似的，且在这个学习过程中，这样选出的子集将包含不同的神经元。



2.4 SOM 网络的 MATLAB 实现

在 MATLAB 神经网络工具箱中提供了若干函数实现 SOM 网络。

1. 创建函数 newsom

该函数用于创建一个 SOM 网络，其调用格式为：

```
net = newsom(P,[D1,D2,...],TFCN,DFCN,STEPS,IN)
```

其中，

P 为一个 $R \times Q$ 维的输入矩阵， R 为输入向量的个数；

[d1, d2,..., di] 为自组织 SOM 网络维数，默认值为[5 8]；

TFCN 为拓扑函数，默认为“hextop”；

DFCN 为距离函数，默认为“linkdist”；

STEPS 为分类阶段的步数，默认为 100；

IN 为初始分类大小，默认值为 3；

net 为生成的 SOM 网络。

【例 2-1】 建立一个输入向量分布在一个二维空间中。

```
>> load simpleclass_dataset
net = newsom(simpleclassInputs,[8 8]);
plotsom(net.layers{1}.positions)
net = train(net,simpleclassInputs);
figure;
plot(simpleclassInputs(1,:),simpleclassInputs(2,:), ...
     'g','markersize',20)
hold on
plotsom(net.iw{1,1},net.layers{1}.distances)
hold off
```

运行程序，其训练过程记录图如图 2-4 所示，二维自组织特征映射神经网络如图 2-5 所示，训练权值分布情况如图 2-6 所示。

2. 传递函数

在 MATLAB 中提供了 softmax 函数实现软最大传递函数，其调用格式为：

```
net= softmax(N,FP)
info = softmax('code')
```

各参数含义参见 compet。与 compet 不同的是，参数 net 为函数返回向量，各元素在区间 [0, 1] 之间，且向量结构与 N 一致。

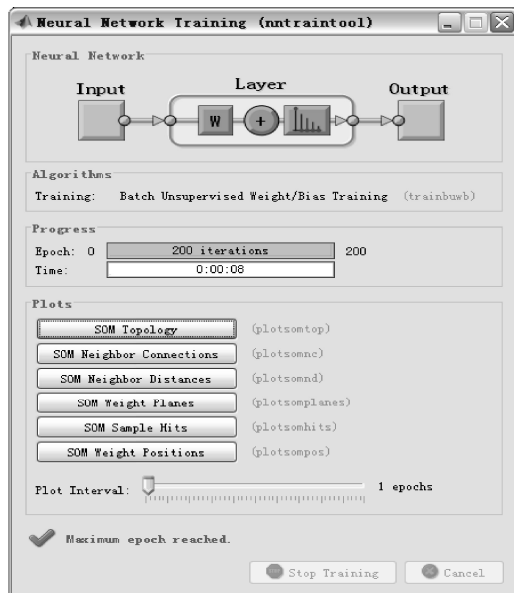


图 2-4 训练过程记录图

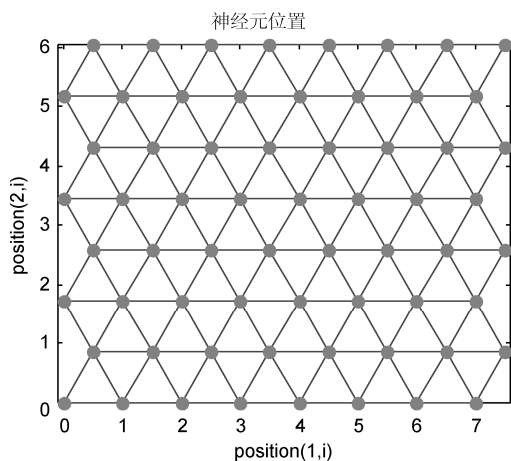


图 2-5 二维自组织特征映射神经网络

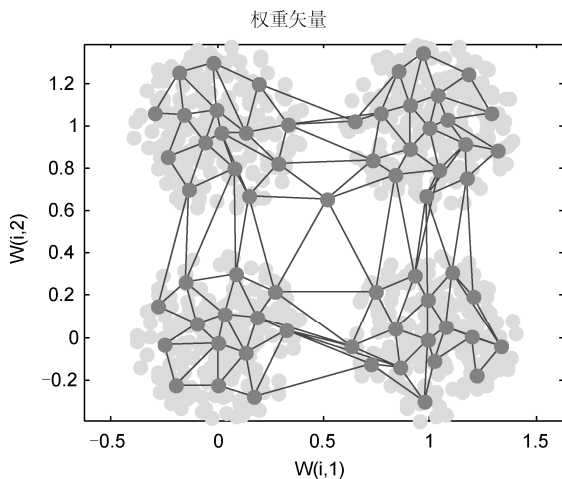


图 2-6 训练权值分布情况

【例 2-2】 利用最大传递函数求给定向量的返回值。

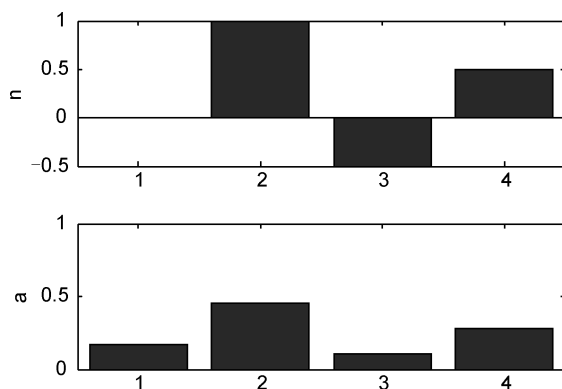
```
>> clear all;
n = [0; 1; -0.5; 0.5];
a = softmax(n)
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

运行程序，输出如下，效果如图 2-7 所示。

```
a =
    0.1674
```



0.4551
0.1015
0.2760

图 2-7 softmax 函数的向量 n 与 a

3. 距离函数

在 MATLAB 神经网络工具箱中提供了若干函数实现 SOM 网络神经元间的距离。

1) boxdist 函数

该函数为 Box 距离函数。在给定神经网络某层神经元的位置后，可利用该函数计算神经元之间的距离。该函数通常用于结构函数的 `gridtop` 的神经网络层，其调用格式为：

```
d=boxdist(pos)
```

其中，`pos` 为神经元位置的 $N \times S$ 维矩阵；

`d` 为函数返回值，神经元距离的 $S \times S$ 维矩阵。

函数的运算原理为 $d(i, j) = \max \|P_i - P_j\|$ ，其中， $d(i, j)$ 表示距离矩阵中的元素； P_i 表示位置矩阵的第 i 列向量。

【例 2-3】 求创建的随机矩阵的 Box 距离。

```
>> clear all;
pos = rand(3,6);
d = boxdist(pos)
```

运行程序，输出如下：

```
d =
    0    0.6599    0.6733    0.6716    0.7634    0.4138
    0.6599    0    0.6487    0.4419    0.7193    0.4436
    0.6733    0.6487    0    0.6604    0.7633    0.3039
    0.6716    0.4419    0.6604    0    0.7311    0.4553
    0.7634    0.7193    0.7633    0.7311    0    0.4594
    0.4138    0.4436    0.3039    0.4553    0.4594    0
```



2) linkdist 函数

该函数为连接距离函数。在给定神经元的位置后，该函数可用于计算神经元之间的距离，其调用格式为：

```
d = linkdist(pos)
```

其中，

pos 为 $N \times S$ 维的神经元位置矩阵；

d 为 $S \times S$ 维的距离矩阵。

函数的原理为：

$$d(i, j) = \begin{cases} 0, & \text{如果 } i = j \\ 1, & \text{如果 } \text{sum}((P_i - P_j)^2)^{\frac{1}{2}} \leq 1 \\ 2, & \text{如果存在 } k, \text{ 使得 } d(i, k) = d(k, j) = 1 \\ 3, & \text{如果存在 } k_1, k_2, \text{ 使得 } d(i, k_1) = d(k_1, k_2) = d(k_2, j) = 1 \\ N, & \text{如果存在 } k_1, k_2, \dots, k_N, \text{ 使得 } d(i, k_1) = d(k_1, k_2) = \dots = d(k_N, j) = 1 \\ S, & \text{其他} \end{cases}$$

【例 2-4】 求创建随机矩阵神经元间的连接距离。

```
>> clear all;
pos = rand(3,6);
D = linkdist(pos)
```

运行程序，输出如下：

```
D =
    0     1     1     1     1     1
    1     0     1     1     1     1
    1     1     0     1     1     1
    1     1     1     0     1     1
    1     1     1     1     0     1
    1     1     1     1     1     0
```

3) mandist 函数

该函数为计算 Manhattan 距离的权函数，其调用格式为：

```
mandist(X Y)
```

计算 X 中每个行向量与 Y 中每个列向量之间的绝对距离，X 的行向量维数必须等于 Y 的列向量维数。

各参数含义参见 dist 函数。

函数的运算原理为：

$$d = \text{sum}(\text{abs}(X - Y))$$

其中，X 和 Y 为两个向量。

【例 2-5】 求创建随机矩阵 Manhattan 的距离。

```
>> clear all;
```



```
W = rand(4,3);  
P = rand(3,1);  
Z = mandist(W,P)
```

运行程序，输出如下：

```
Z =  
    1.0139  
    0.6981  
    0.9628  
    0.6889
```

4. 学习函数 learnsom

该函数为自组织映射的权值学习函数，其调用格式为：

```
[dW,LS] = learnsom(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)  
info = learnsom('code')
```

各参数含义参见 learnk。

在利用该函数进行学习前，需要设置如表 2-1 所示的学习参数。

表 2-1 学习参数

函数名称	默认值	属性
LP.order_lr	0.9	分类阶段学习速率
LP.order_steps	1000	学习阶段步长
LP.tune_lr	0.02	调谐阶段领域距离
LP.tune_nd	1	调谐阶段学习速率

以上各值都是 MATLAB 的默认值。如果需要调整，只要按照上面的格式重新设定即可。

【例 2-6】 对所创建的 SOM 网络进行学习。

```
>> clear all;  
p = rand(2,1);  
a = rand(6,1);  
w = rand(6,2);  
pos = hextop(2,3);           % 创建含有 12 个神经元的自组织 SOM 网络  
d = linkdist(pos);           % 计算各神经元的连接距离  
% 设置学习的参数属性  
lp.order_lr = 0.9;  
lp.order_steps = 1000;  
lp.tune_lr = 0.02;  
lp.tune_nd = 1;  
ls = [];                     % 学习速率为空  
[dW,ls] = learnsom(w,p,[],[],a,[],[],[],d,lp,ls)
```

运行程序，输出如下：



```
dW =
    0.4964    0.8594
    1.5497    0.0337
    1.3739    0.2049
    0.5942    1.1356
    0.1022    0.4911
   -0.1869    0.7662

ls =
    step: 1
    nd_max: 2
```

5. 初始化函数 midpoint

该函数为中点权值初始化函数，其调用格式为：

```
W = midpoint(S,PR)
```

其中，

S 为神经元的数目；

PR 为输入向量取值范围的矩阵；

W 为函数返回权值矩阵。

6. 权值函数 negdist

该函数为负距离权值函数，其调用格式为：

```
Z = negdist(W,P)
```

其中，

W 为 $S \times R$ 维的权值矩阵；

P 为 Q 组输入向量的 $R \times Q$ 维矩阵；

df=negdist('deriv')：返回值为空，因为该函数不存在导函数。

该函数的运算原理为：

$$Z = -\sqrt{\sum (w - p)^2}$$

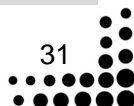
实际上这也是负欧氏距离。

【例 2-7】 求创建随机矩阵的负欧氏距离。

```
>> W = rand(4,3);
P = rand(3,1);
Z = negdist(W,P)
```

运行程序，输出如下：

```
Z =
   -0.4715
   -0.7181
   -0.9749
   -1.0970
```



7. 显示函数 plotsom

该函数用于绘制自组织特征映射，其调用格式为：

```
plotsom(pos)
plotsom(W,D,ND)
```

其中，

pos 为 S 个 N 维神经元的位置向量；

W 为权值矩阵；

D 为距离矩阵；

ND 为邻域矩阵，默认为 1。

plotsom(pos) 利用红点绘制神经元的位置，将欧氏距离小于或等于 1 的神经元连接起来；plotsom(W,D,ND) 将欧氏距离小于或等于 1 的神经元的权值向量连接起来。

其用法可参考例 2-5。

8. 结构函数

在 MATLAB 神经网络工具箱中提供了若干结构函数，实现自组织网络的结构化。

1) hextop 函数

该函数为六角层结构函数，其调用格式为：

```
pos=hextop(dim1,dim2,...,dimN)
```

其中，

dim i 为维数为 i 的层的长度；

pos 为由 N 个并列向量组成的 $N \times S$ 维矩阵，其中， $S = \text{dim1} \times \text{dim2} \times \cdots \times \text{dimN}$ 。

2) gridtop 函数

该函数为网格层结构函数，其调用格式为：

```
gridtop(dim1,dim2,...,dimN)
```

各参数含义参见 hextop 函数。

3) randtop 函数

该函数为随机层结构函数，其调用格式为：

```
pos = randtop(dim1,dim2,...,dimN)
```

各参数含义参见 hextop 函数

【例 2-8】 利用这 3 个结构函数创建一个二维的神经网络层。

```
>> clear all;
pos1 = gridtop(8,5);
subplot(221);plotsom(pos1);
title('网格层结构函数')
```




```
pos2 = hextop(8,5);
subplot(222);plotsom(pos2)
title('六角层结构函数')
pos3 = randtop(8,5);
subplot(223);plotsom(pos3)
title('随机层结构函数')
```

运行程序，效果如图 2-8 所示。

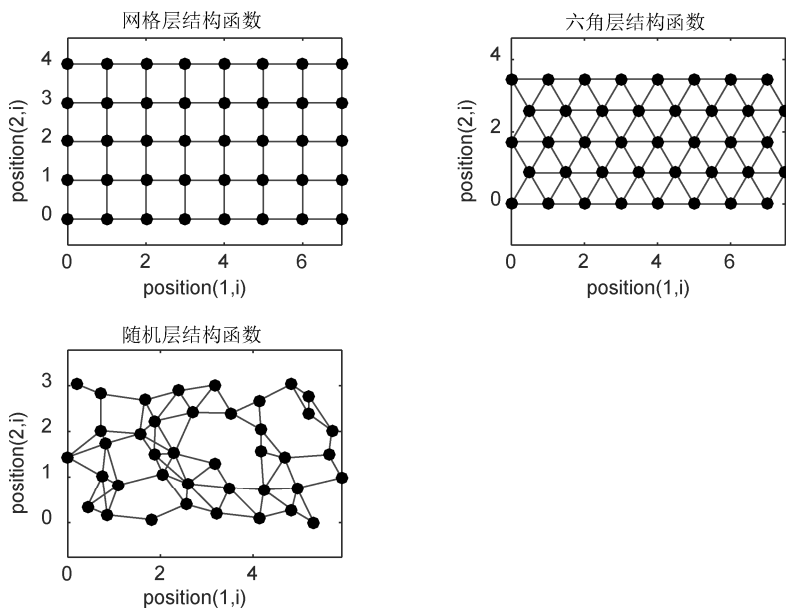


图 2-8 3 个结构函数产生的 40 个神经元的分布位置效果

2.5 SOM 网络的应用

下面利用示例说明 SOM 网络在实际领域中的应用。

1. SOM 网络在分类中的应用

【例 2-9】 针对给定的输入向量 P ，建立一个自组织特征映射，并对 P 进行分类。首先建立一个 SOM 网络，并绘制出网络当前神经元的位置。

```
P=[0.1 0.8 0.1 0.9;0.2 0.9 0.1 0.8];
net=newsom([0 2;0 1],[3 5]);
plotsom(net.layers{1}.positions)
title('神经元的位置')
```

创建的 SOM 网络竞争层为一个二维的 3×5 的平面阵列，拓扑函数为 `hextop`，距离函数为 `linkdist`，其他参数均取默认值。

此时的神经元位置的初始分布如图 2-9 所示。由图可见，此时的神经元位置是均匀分布的，也就是说，网络还没有对输入向量进行分类的能力。

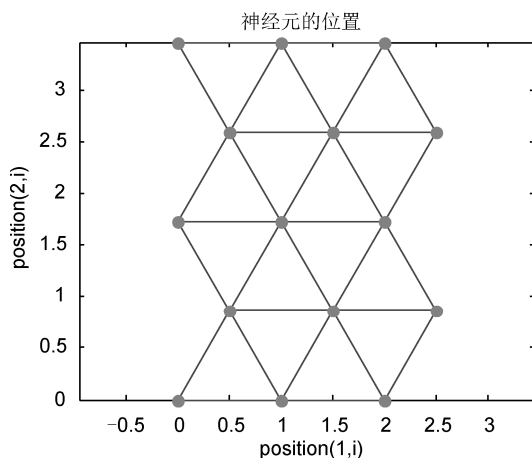


图 2-9 神经元位置的初始分布

接下来对网络进行训练，得到自组织网络的训练过程如图 2-10 所示。检查训练过程中，神经元位置的变化分别如图 2-11 和图 2-12 所示。

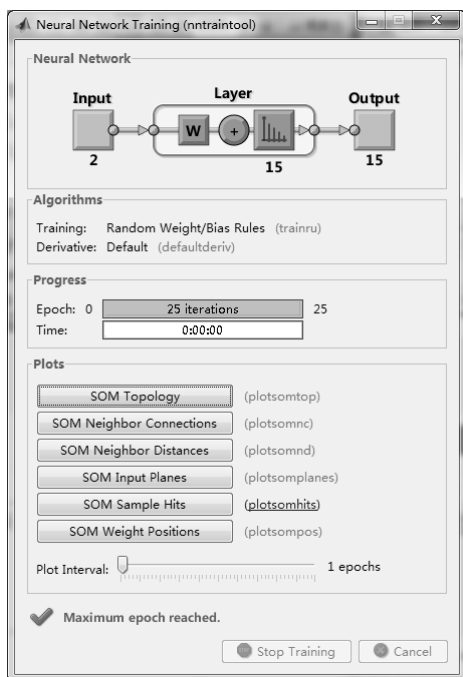


图 2-10 自组织网络的训练过程

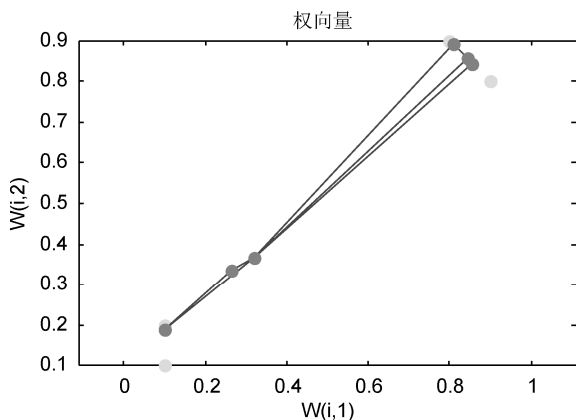


图 2-11 神经元位置（训练次数：10）

本例的 MATLAB 代码为：

```
P=[0.1 0.8 0.1 0.9;0.2 0.9 0.1 0.8];
net=newsom([0 2;0 1],[3 5]);
```



```

plotsom(net.layers{1}.positions)
title('神经元的位置')
%进行训练
%训练次数为 10
net.trainParam.epochs=10;
net=train(net,P);
plot(P(1,:),P(2,:),'.g','markersize',20);
hold on;
%绘制训练后神经元的位置
plotsom(net.iw{1,1},net.layers{1}.distances);
hold off;
title('权向量');
figure;
%训练次数为 25
%训练前进行初始化
net=init(net);
net.trainParam.epochs=25;
net=train(net,P);
plot(P(1,:),P(2,:),'.g','markersize',20);
hold on;
plotsom(net.iw{1,1},net.layers{1}.distances);
hold off;
title('权向量');

```

由图 2-11 可见, 经过 10 次训练后, 神经元的位置就发生了明显的改变。神经元位置的分布状况表示它们已经可以对输入向量进行分类了。此时, 再增加训练次数已经没有实际意义了。经过 25 次训练后的网络神经元分布和 10 次训练后的神经元分布没有明显的差异, 如图 2-12 所示。

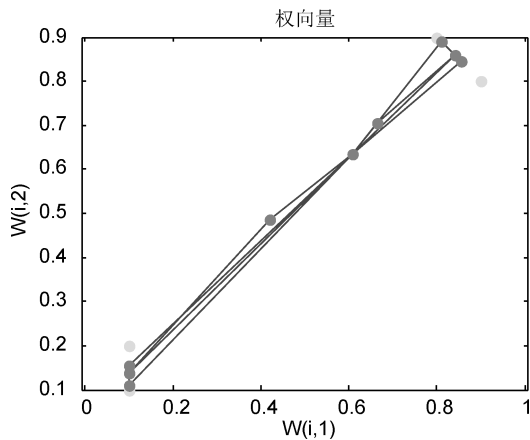


图 2-12 神经元位置 (训练次数: 25)

经过 10 次训练后，网络的输出为：

```
Yc =
     2     1     2     1
```

由此可见，SOM 网络的分类结果比较精细，把 4 个点分为了 4 类，即激发了 1、13、2、14 四个神经元。但是，通过对结果进行分析可以发现，在上例中处于同一类的点在这里激发的神经元位置也是邻近的，这说明相对于基本竞争型网络来说，SOM 网络的分类结果更加准确。

经过 25 次训练后，网络的输出值与此类似。需要注意的是，重新运行上述代码，可能结果就会不一致，这里因为每次激发的神经元不一样。但是，相似的类激发的神经元总是邻近的，差别很大的类激发的神经元相差也比较远。

【例 2-10】 人口分类是人口统计中的一个重要指标。由于各方面的原因，我国人口的出生率在性别上的差异比较大，具体表现在同一个时期出生的人口中，一般男的占多数，大大超过了正常的比例。因此，正确地进行人口分类是制定合理人口政策的基础。

通过分析历史资料，得到了在 1999 年 12 月共 20 个地区的人口出生比例情况，如表 2-2 所示。

表 2-2 人口出生比例

男 (%)	0.5512	0.5123	0.5087	0.5001	0.6012	0.5298	0.5000	0.4965	0.5103	0.5003
女 (%)	0.4488	0.4877	0.4913	0.4999	0.3988	0.4702	0.5000	0.5035	0.4897	0.4997

将上表中的数据作为网络的输入样本 P。P 是一个二维随机向量，它的分布情况如图 2-13 所示。

```
>> P=[0.5512 0.5123 0.5087 0.5001 0.6012 0.5298 0.5000 0.4965 0.5103 0.5003;
      0.4488 0.4877 0.4913 0.4999 0.3988 0.4702 0.5000 0.5035 0.4897 0.4997];
plot(P(1,:),P(2,:),'*r');
hold on
```

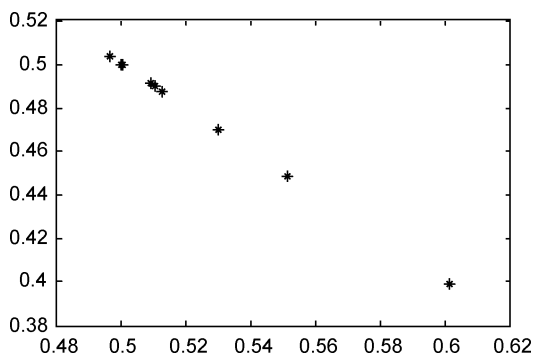


图 2-13 样本数据的分布

利用 12 个神经元的 SOM 网络对输入向量 P 进行分类。该网络竞争层神经元的组织结构为 3×4，通过距离函数 linkdist 来计算距离，网络创建代码如下。



```
net=newsom([0 1;0 1],[3 4]);
w1_init=net.iw{1,1};
plotsom(w1_init,net.layers{1}.distances);
title('权向量');
```

运行结果如图 2-14 所示，图中每一点表示一个神经元，由于网络的初始权值都被设置为 0.5，所以这些点在图中是重合的，看起来就像一个点，实际上是 12 个点。

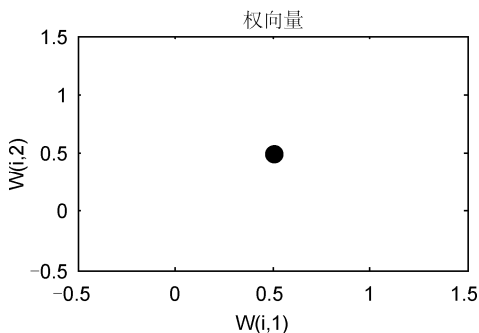


图 2-14 网络初始权值的分布

在命令窗口中查看 w1_init 的值，可得：

```
w1_init =
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
    0.5000    0.5000
```

接下来利用训练函数 **train** 对网络进行训练，设想经过训练的网络可对输入向量进行正确分类。网络训练步数对于网络性能的影响比较大，所以这里将步数设置为 100、300 和 500，并分别观察其权值分布。

步数为 100 时的权值分布如图 2-15 所示。训练步数为 100 时的训练代码为：

```
net=train(net,P);
figure;
w1=net.iw{1,1};
plotsom(w1,net.layers{1}.distances)
title('权向量');
```

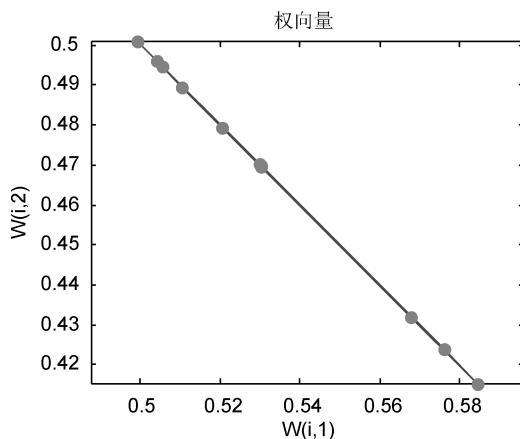


图 2-15 权值分布（训练步数为 100）

步数为 300 时的权值分布如图 2-16 所示。

```
% 训练步数为 300 时的训练代码
net.trainParam.epochs=300;
net=init(net);
net=train(net,P);
figure;
w1=net.iw{1,1};
plotsom(w1,net.layers{1}.distances)
```

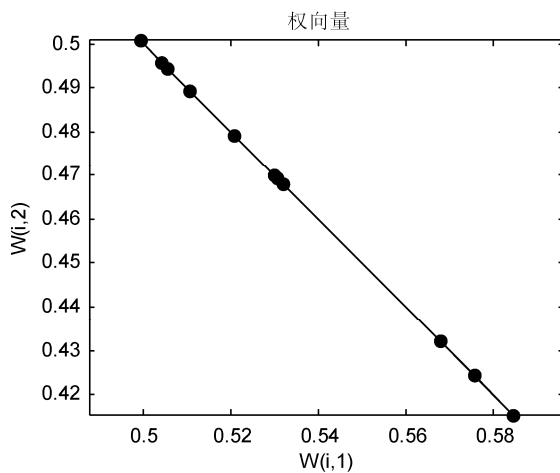


图 2-16 权值分布（训练步数为 300）

步数为 500 时的权值分布如图 2-17 所示。

```
% 训练步数为 500 时的训练代码
net.trainParam.epochs=500;
net=init(net);
net=train(net,P);
figure;
```



```
w1=net.iw{1,1};
plotsom(w1,net.layers{1}.distances)
title('权向量');
```

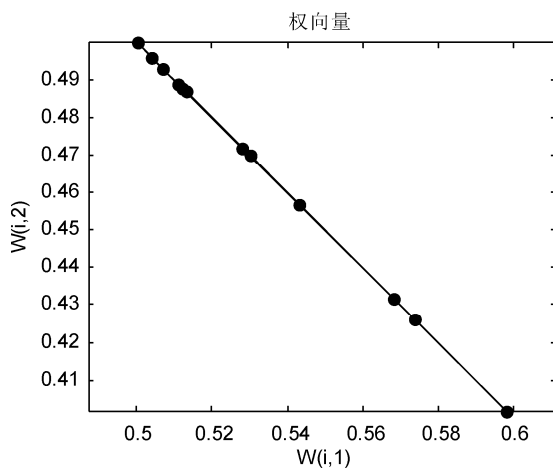


图 2-17 权值分布（训练步数为 500）

从图 2-15、图 2-16 和图 2-17 可以看出，训练了 100 步以后，神经元就开始自组织地分布了，每个神经元可以区分不同的样本。随着训练步数的增多，神经元的分布更加合理，但是，当训练次数达到一定值后，权值分布的改变就不明显了。比如，训练 300 步和训练 500 步后的权值分布就比较相似。

网络训练结束后，权值也就固定了。以后每输入一个值，网络就会自动地对其进行分类。因此，利用这一点对网络进行测试。首先，利用仿真函数 `sim` 来观察网络对样本数据的分类结果。

```
y=sim(net,P);
Y=vec2ind(y)
```

输出结果为：

```
Y =
     4     10     10     12     1     6     12     12     10     12
```

对结果进行分析，如表 2-3 所示。

表 2-3 聚类结果

样 本 序 号	类 别	激发神经元的索引
1	1	4
2	2	3
3	3	10
4 7 10	4	11
5	5	1

续表

样 本 序 号	类 别	激发神经元的索引
6	6	7
8	7	12
9	8	6

现在，输入一个某地的出生性别比例，检验它属于哪一类。

```
p=[0.5;0.5];
y=sim(net,p);
y=vec2ind(y)
```

结果为 11。由此可见，此时激发了网络的第 11 个神经元，所以 p 属于第 4 类。通过直接对比数据可知， p 确实与样本中的第 4 组、第 7 组和第 10 组数据非常接近。

2. SOM 网络应用于故障诊断

【例 2-11】 给出一个含有 8 个故障样本的数据集。每个故障样本中有 8 个特征，分别为最大压力 (P_1)、次最大压力 (P_2)、波形幅度 (P_3)、上升沿宽度 (P_4)、波形宽度 (P_5)、最大余波的宽度 (P_6)、波形的面积 (P_7)、起喷压力 (P_8)，使用 SOM 网络进行故障诊断。故障样本如表 2-4 所列（数据已归一化）。

表 2-4 常见的 8 种故障特征

输入 样 本								
故 障 原 因	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
T_1	0.9325	1.0000	1.0000	-0.4526	0.3895	1.0000	1.0000	1.0000
T_2	-0.4571	-0.2854	-0.9024	-0.9121	-0.0841	1.0000	-0.2871	0.5647
T_3	0.5134	0.9413	0.9711	-0.4187	0.2855	0.8546	0.9478	0.9512
T_4	0.1545	0.1564	-0.5000	-0.6571	-0.3333	-0.6667	-0.3333	-0.5000
T_5	0.1765	0.7648	0.4259	-0.6472	-0.0563	0.1726	0.5151	0.4212
T_6	-0.6744	-0.4541	-0.8454	1.0000	-0.8614	-0.6714	-0.6279	-0.6785
T_7	0.4647	0.8710	0.0712	-0.7845	-0.2871	0.8915	0.6553	0.6152
T_8	0.6818	1.0000	-0.625	-0.8426	-0.6215	-0.1574	1.0000	0.7782

应用 SOM 神经网络诊断柴油机故障的步骤如下：

- (1) 选取标准故障样本。
- (2) 对每一种标准故障样本进行学习，学习结束后，对具有最大输出的神经元标以该故障的记号。
- (3) 将待检样本输入 SOM 神经网络中。
- (4) 如果输出神经元在输出层的位置与某标准故障样本的位置相同，说明待检样本发生了相应的故障；如果输出神经元在输出层的位置介于很多标准故障之间，说明这几种标准故障都有可能发生，且各故障的程度由该位置与相应标准故障样本位置的欧氏距离确定。

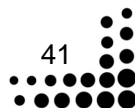


实现的 MATLAB 代码为:

```
>> clear all;
%样本数据
p=[ 0.9325    1.0000    1.0000   -0.4526    0.3895    1.0000    1.0000    1.0000;...
   -0.4571   -0.2854   -0.9024   -0.9121   -0.0841    1.0000   -0.2871    0.5647;...
    0.5134    0.9413    0.9711   -0.4187    0.2855    0.8546    0.9478    0.9512;...
    0.1545    0.1564   -0.5000   -0.6571   -0.3333   -0.6667   -0.3333   -0.5000;...
    0.1765    0.7648    0.4259   -0.6472   -0.0563    0.1726    0.5151    0.4212;...
   -0.6744   -0.4541   -0.8454    1.0000   -0.8614   -0.6714   -0.6279   -0.6785;...
    0.4647    0.8710    0.0712   -0.7845   -0.2871    0.8915    0.6553    0.6152;...
    0.6818    1.0000   -0.625   -0.8426   -0.6215   -0.1574    1.0000    0.7782];
%newsom 建立 SOM 网络。Minmax(p)取输入的最大、最小值。竞争层为 6×6=36 个神经元
net=newsom(minmax(p),[6 6]);
plotsom(net.layers{1}.positions)
%7 次训练的次数
a=[10 30 50 100 200 500 1000];
%随机初始化一个 7×8 向量
yc=rands(7,8);
%训练次数为 10 次
net.trainparam.epochs=a(1);
%训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(1,:)=vec2ind(y);
plotsom(net.iw{1,1},net.layers{1}.distances)

%训练次数为 30 次
net.trainparam.epochs=a(2);
%训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(2,:)=vec2ind(y);
plotsom(net.iw{1,1},net.layers{1}.distances)

%训练次数为 50 次
net.trainparam.epochs=a(3);
%训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(3,:)=vec2ind(y);
plotsom(net.iw{1,1},net.layers{1}.distances)
```





```
% 训练次数为 100 次
net.trainparam.epochs=a(4);
% 训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(4,:)=vec2ind(y);
plotsom(net.iw{1,1},net.layers{1}.distances)
```

```
% 训练次数为 200 次
net.trainparam.epochs=a(5);
% 训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(5,:)=vec2ind(y);
plotsom(net.iw{1,1},net.layers{1}.distances)
```

```
% 训练次数为 500 次
net.trainparam.epochs=a(6);
% 训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(6,:)=vec2ind(y);
plotsom(net.iw{1,1},net.layers{1}.distances)
```

% 训练次数为 1000 次，得到 1000 次权值向量图，如图 2-18 所示。

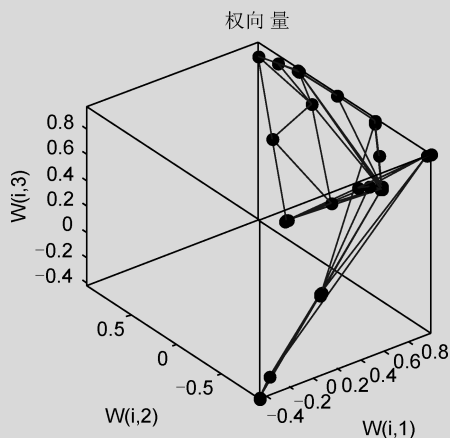


图 2-18 训练为 1000 次的权值向量图

```
net.trainparam.epochs=a(7);
% 训练网络和查看分类
net=train(net,p);
y=sim(net,p);
yc(7,:)=vec2ind(y);
```



```

plotsom(net.iw{1,1},net.layers{1}.distances)
title('权向量')
yc
%网络分类的预测
%测试样本输入
t=[0.9512 1.0000 0.9458 -0.4215 0.4218 0.9511 0.9645 0.8941]';
%用 sim 做网络仿真
r=sim(net,t);
%变换函数，将单值向量转换为下标向量
rr=vec2ind(r)
%查看网络拓扑学结构
plotsomtop(net)

```

%SOM 网络拓扑学结构如图 2-19 所示

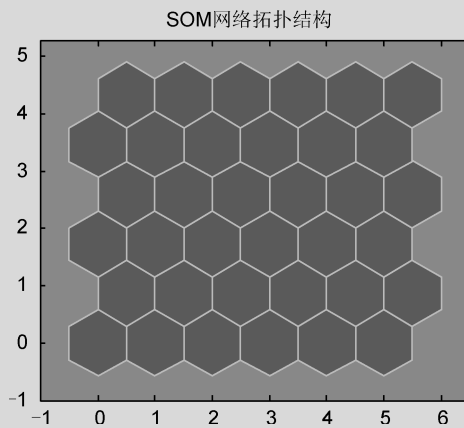


图 2-19 SOM 网络拓扑学结构

```

title('SOM 网络拓扑结构')
>> %查看邻近神经元直接的距离情况
plotsomnd(net)

```

%得到邻近神经元之间的距离效果，如图 2-20 所示

```

title('邻近神经元之间的距离')

```

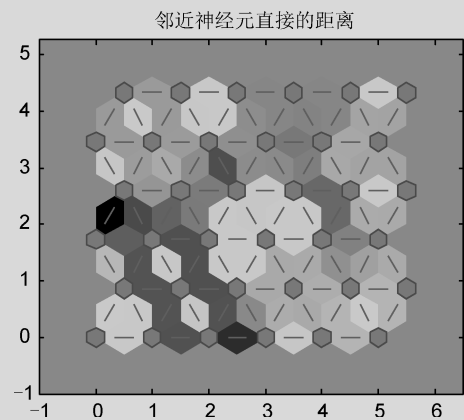
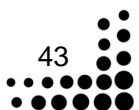


图 2-20 邻近神经元之间的距离效果



>> %查看每个神经元的分类情况

plotsomhits(net,p)

%每个神经元分类情况如图 2-21 所示

title('每个神经元的分类')

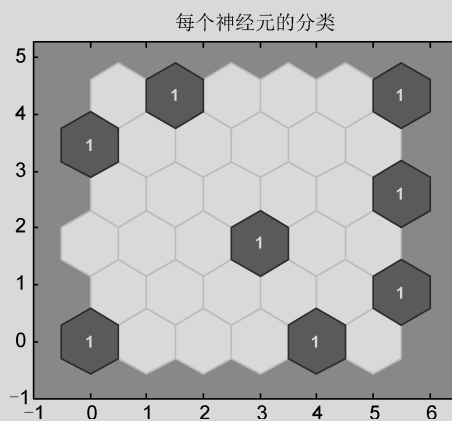


图 2-21 每个神经元的分类情况

聚类的结果如 yc 所示：当训练步数为 10 时，故障原因：1、5 分为一类，2、6、7、8 分为一类，3 单独为一类、5 单独为一类。可见，网络已经对样本进行了初步的分类，这种分类不够精准。

yc =

3	1	2	36	3	1	1	1
36	36	7	1	2	30	36	36
8	24	2	1	31	36	24	24
21	36	31	1	25	4	30	18
21	36	31	1	19	4	23	12
28	36	31	1	9	6	30	18
16	5	25	1	32	36	12	24

rr =

24

当训练步数为 100 时，每个样本都被划分为一类。这种分类结果更加细化了。当训练步数为 200、500 或者 1000 时，同样是每个样本都被划分为一类。这时，如果再提高训练步数，已经没有实际意义了。

由图 2-19 可知，竞争层神经元有 $6 \times 6 = 36$ 个；在图 2-20 中，蓝色代表神经元，红色线代表神经元直接连接，每个菱形中的颜色表示神经元之间的距离，从黄色到黑色，颜色越深说明神经元之间的距离越远。图 2-21 中蓝色神经元表示竞争型的神经元。

从图 2-21 可以看出，SOM 网络将未知故障样本分到了第一类故障中。

第3章 线性网络的实际应用

3.1 线性化建模

由线性网络的输入/输出关系式 $A=WP+B$ 来看, 如果将偏差的输入 1 归结为 P 中的一个输入, 可得增广权值, 则关系式可以写成 $A=WP$ 。只要 P 不奇异, 逆存在, 就能求出使得关系式成立的精确值。一般情况下, 线性网络能够给出满足给定误差下的网络权值, 这个求解过程就是设计一个线性网络, 使其外部的输入/输出特性等价于从实际系统中获得的样本对特定的系统建模任务。从以上的学习来看, 这似乎是一件很容易的事: 可以精确建模的用 `newlind` 即可获得, 带有误差的使用 `newlin` 和 `train` 两个函数即可设计出网络的权值。但在实际系统中, 当给定从实际系统中获得的一组样本输入到输出的转换, 这涉及原系统输出与输入之间的函数关系式的问题。只有把这个问题搞懂了, 对建模与应用的问题才有可能处理正确。

任何实际系统都是由物理元器件组成的, 系统建模一般是建立在物理、化学等原理基础上的, 所建立出的输出与输出之间的关系式一般具有微积分形式。自动控制原理中采用拉普拉斯变换, 将微分由 s 表示, 积分由 $\frac{1}{s}$ 表示, 从而将模型的微积分表达式变换为代数表达式, 简化了模型, 方便了对模型中参数的求解。根据已知条件对模型中参数的求解就是参数辨识。另外一种系统是建模, 它是通过实验手段测得系统的输入/输出数据, 然后求解系统的最佳匹配模型, 这种用实验方法获得的数据一般采用的数据是离散的, 由此获得的模型是 Z 变换模型。形式不再是微积分的, 而是差分的。

最简单的系统模型 (或函数) 的数学表达式为:

$$y = f(x) \text{ 或 } y(k) = f(x(k)) \quad (3-1)$$

而实际物理系统的输入/输出关系式往往都比较复杂, 一般的形式为:

$$y(k) = f(x(k-n), x(k-n+1), \dots, x(k)) \quad (3-2)$$

虽然整个系统的外部只有一个 $x(k)$ 的输入, 但由于系统内部元器件之间的作用, 使得系统输出可能是输入的一个较复杂函数。所以设计者在用神经网络进行建模应用时, 最关键的是要选择好输入/输出节点 r 和 s 。神经网络系统建模结构图如图 3-1 所示。

下面通过示例来演示神经网络系统的建模。

【例 3-1】 设计并训练一个线性网络, 实现从输入向量 P 到输出向量 T 的转换。

```
>> clear all;  
P=[1 2 4;2 4 8];  
T=[0.5 1 -1];
```

```
net=newlin(minmax(P),1,0,0.01);
y=sim(net,P)
net=init(net);
net.trainParam.epochs=2000;
net.trainParam.goal=0.0001;
net=train(net,P,T);
```

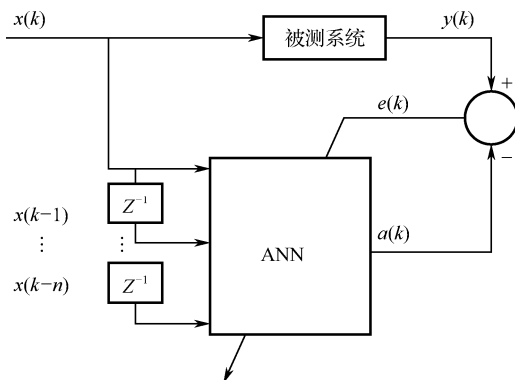


图 3-1 神经网络系统建模结构图

运行程序，网络的训练过程如图 3-2 所示。

对网络进行仿真，得到训练后的网络输出 Y。

```
>> Y=sim(net,P)
Y =
    0.9286    0.3571   -0.7857
```

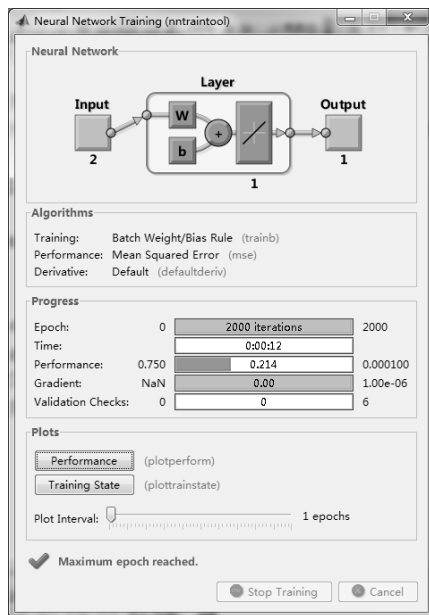


图 3-2 网络训练过程

将 Y 与目标向量 T 相比较，发现误差还是比较大的。说明在这种情况下，网络是无法给出完美解决方案的。

【例 3-2】 设计一个简单的单层线性神经元，其输入和目标分别为 $p=[0.5 \ 36]$ ； $t=[15.9 \ 12.1]$ 。

```
>> clear all;
p=[0.5 36];
t=[15.9 12.1];
w_rang=-2:0.2:2; %权值范围
b_rang=-1:0.1:1; %阈值范围
es=errsurf(p,t,w_rang,b_rang,'purelin'); %误差曲面
plotes(w_rang,b_rang,es);
net=newlin(p,t); %设置一个单层线性神经网络
a=0; e=0; sse=0; %参数置零
a=sim(net,p) %对网络进行验证
e=t-a %求误差
sse=sumsq(e);
plotep(net.iw{1,1},net.b{1},sse);
```

运行程序，输出如下，效果如图 3-3 所示。

```
a =
    15.9000    12.1000
e =
    1.0e-014 *
         0         0.1776
```

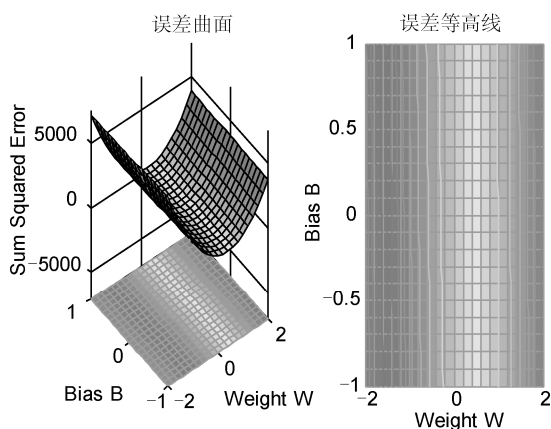


图 3-3 权值与阈值的误差曲面及误差等高线图

下面设计一个线性网络并对此进行训练以求最优解。

```
>> maxlr=0.4*maxlinlr(p,'bias'); %设置学习速率
net=newlin([0.5 6],1,[0],maxlr); %设计一个线性神经网络
net.trainParam.goal=0.001; %训练目标
```

```
net.trainParam.epochs=500;           %训练步数
[net,tr]=train(net,p,t);             %训练
a=sim(net,p)                         %仿真
```

运行程序，输出如下，效果如图 3-4 所示。

```
a =
    2.3673    12.2984
```

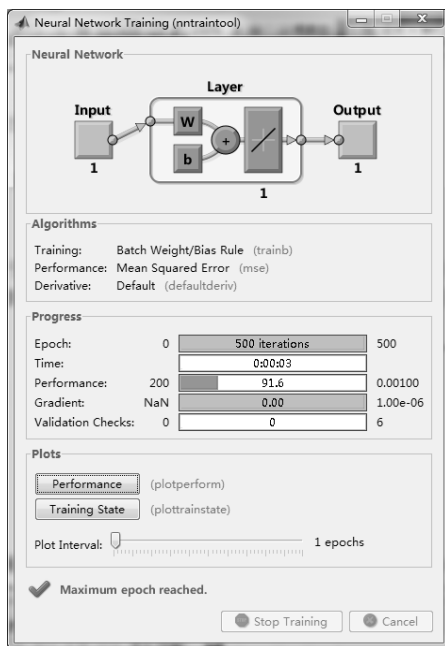


图 3-4 网络训练效果

【例 3-3】 给定如下的输入向量 **P** 和输出向量 **T**，再次利用线性网络求解两者之间的关系（本例题在于了解线性网络的线性逼近求解的能力）。

```
>>P=[1 1.5 3.0 -1.2];
T=[0.5 1.1 3.0 -1.0];
%绘制训练向量与目标向量效果图
plot(P,T,'o');                                     %效果如图 3-5 所示
axis([-1.5 3.5 -1.5 3.5]);
xlabel('训练向量 P');
ylabel('目标向量 T');

%创建一个线性网络，并进行训练
net=newlin(minmax(P),1,0,0.01);
net=init(net);
net.trainParam.epochs=500;
net.trainParam.goal=0.0001;
net=train(net,P,T);
```




运行程序，得到训练效果如图 3-6 所示。

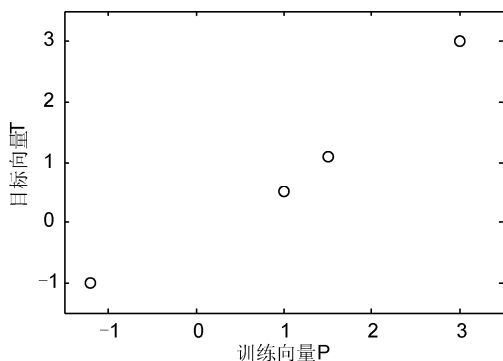


图 3-5 输入向量 P 与目标向量 T 效果图

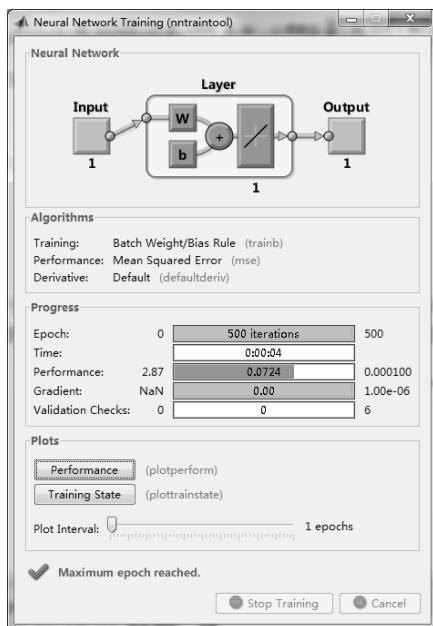


图 3-6 网络训练效果

```
%检验网络训练误差，对网络进行仿真
>> y=sim(net,P)
y =
    0.8299    1.2975    2.7003   -1.2276
```

由结果可知，网络的训练误差比较大，接着检验网络是否很好地逼近 P 与 T 之间的线性关系。

```
>> plot(P,y);
hold on;
plot(P,T,'r+')
```

运行程序，得到网络逼近曲线（如图 3-7 所示）。

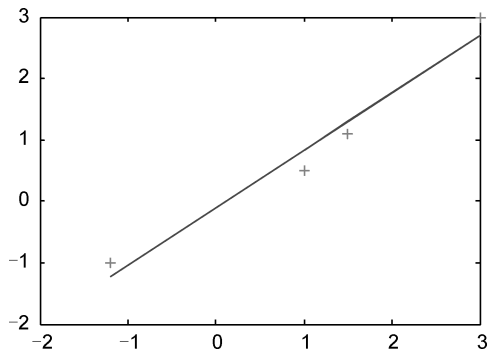


图 3-7 网络逼近曲线



由图 3-7 可看出,网络的逼近误差是比较大的。因此,可以得出如下结论:线性网络只可以学习输入/输出向量之间的线性关系。所以,对于某些特殊的问题,网络无法得到满意的结果。但是,即使不存在一个完美的结果,只要学习速率足够小,对于给定的结构,线性网络总可得到一个接近目标的结果。

3.2 模式分类

除了前面介绍的感知器可应用于模式分类外,线性网络也可应用于模式分类。

【例 3-4】 以单层线性网络模拟与函数,其函数真值表如表 3-1 所示。

表 3-1 与函数真值表

输入 $p1p2$	输出 a
0 0	0
1 0	0
0 1	0
1 1	1

若把与函数看成 $p1$ - $p2$ 平面上的点,则点 $A0(0,0)$, $A1(0,1)$ 和 $A2(1,0)$ 表示输出为 0 的 3 个点, $B0(1,1)$ 表示输出为 1 的一个点,如图 3-8 所示。

可以看出,与函数是一个简单的线性划分问题,用一个线性神经元构成的网络就可以实现。

根据以上分析,按本题要求设计的线性神经网络的基本结构为:

- 网络有 1 个输入向量,包括 2 个元素,输入元素的取值范围为[0 1];
- 输出向量有一个元素,为二值变量 0 或 1。

设计的线性神经网络结构示意图如图 3-9 所示。

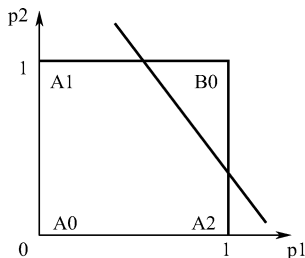


图 3-8 与函数的图形表示

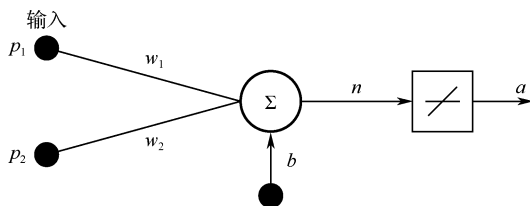


图 3-9 线性神经网络结构示意图

```
>>clear all; %清除所有内存变量
%设计线性神经网络
p=[0 0;0 1;1 0;1 1]; %输入向量
t=[0 0 0 1]; %目标向量
net=newlind(p,t); %设计线性神经网络
```



```
w=net.IW{1};           %输出训练后的权值
b=net.b{1}             %输出训练后的阈值
%线性神经网络的仿真
a=sim(net,p)           %输出仿真结果
y=a>0.5                %将模拟仿真结果转换为数字量
```

运行程序，输出如下：

```
b =
    -0.2500
a =
   -0.2500    0.2500    0.2500    0.7500
y =
     0     0     0     1
```

3.3 消噪处理

使用自适应滤波器的线性滤波功能可以进行网络消噪。下面以飞行中飞机机长同乘客之间的广播模型为例来说明其工作原理。在飞行的飞机中，如果飞行员对着麦克风说话，驾驶座舱的引擎噪声将会混入他的声音信号中，这使得旅客听到的是非常嘈杂的声音。我们需要的是飞行员的声音而不是飞机引擎的噪声，如果可以获得引擎噪声样本并把它作为自适应滤波器的输入，那么就可以通过自适应滤波器来消除引擎噪声的影响。消除噪声的自适应滤波器的训练及其工作结构图如图 3-10 所示。

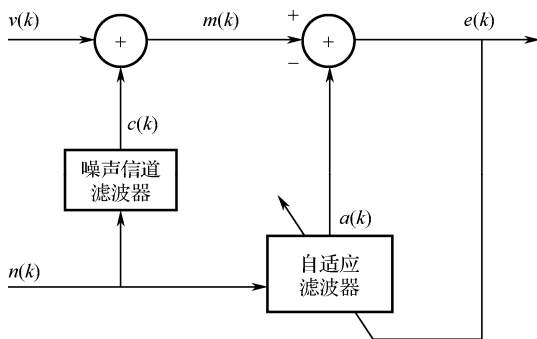


图 3-10 自适应滤波器消噪的训练及其工作结构图

这里采用自适应神经元线性网络去逼近带有引擎噪声信号 $n(k)$ 的飞行员声音信号 $m(k)$ 。由引擎噪声信号 $n(k)$ 给网络提供输入信息，从图 3-10 中可以看到，网络通过自适应滤波器的输出信号去逼近混杂在声音信号中的噪声部分，调整自适应滤波器以便使误差 $e(k)$ 最小。由图 3-10 可得以下关系式：

$$m(k) = v(k) + c(k)$$

$$e(k) = m(k) - a(k)$$

由此可得：



$$e(k) = v(k) + c(k) - a(k)$$

当自适应滤波器成功地逼近信号 $c(k)$ 时, 在得到的 $e(k)$ 信号中就只剩下了需要的机长的声音信号了。

在最初进行网络的训练过程中, 将机长的声音信号置为 0, 并调节网络的权值, 然后将训练完成后获得的网络再以同样的方式接入系统中加以应用。当网络工作时, 由于网络消除了飞行员声音信号中的噪声信号, 使其输出 $e(k)$ 仅为飞行员的声音。

这种去噪方法优于传统滤波器的之处是把噪声从信号中近似完全消去, 而传统的低通滤波器只是通过高频时很小的放大倍数把噪声抑制掉, 所以网络的逼近精度越高, 消噪的效果越好。

下面通过示例来演示自适应滤波消噪处理。

【例 3-5】 对于一个最优的滤波器, 希望通过滤波将信号中的噪声去掉, 这对一般的滤波器很难完全做到。利用自适应线性网络实现噪声抵消的原理框图如图 3-11 所示。

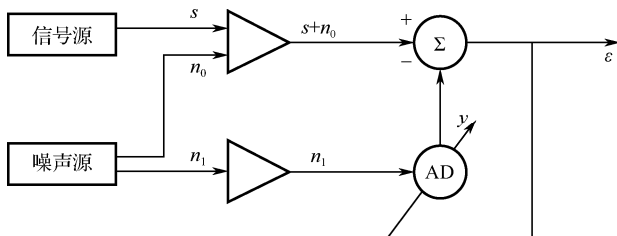


图 3-11 噪声抵消的原理框图

图中 s 为原始输入信号, 假设为平稳的零均值随机信号; n_0 为与 s 不相关的随机噪声; n_1 为与 n_0 相关的信号; 系统输出为 ε ; $s + n_0$ 为 ADALINE 神经元的预期输出, y 为 ADALINE 神经元的输出。则

$$\varepsilon = s + n_0 - y$$

$$\begin{aligned} E[\varepsilon^2] &= E[(s + n_0 - y)^2] = E[s^2] + 2E[s \cdot (n_0 - y)] + E[(n_0 - y)^2] \\ &= E[\varepsilon^2] + E[(n_0 - y)^2] \end{aligned}$$

通过 ADALINE 调节, 得到

$$E_{\min}[s^2] = E_{\min}[s^2] + E_{\min}[(n_0 - y)^2]$$

上式中, 当 $E_{\min}[(n_0 - y)^2] \rightarrow 0$ 时, $y \rightarrow n_0$, 其输出 ε 为 s , 则噪声被抵消。

采用这种系统来完成对胎儿心率的检测, 可以得到十分满意的结果。由于测量胎儿的心率一定会受到母体心率的干扰, 而且母亲心率很强, 但与胎儿心率是相互独立的, 所以可将母体心率作为噪声源 n_1 输入 ADALINE 中, 混有噪声的胎儿心率信号为目标响应, 通过抵消后, 系统就可以得到清晰的胎儿心率。

这种系统还可以应用于电话中的回音抵消。在电话通话的过程中, 如果没有回音抵消措施, 那么, 我们自身的声音会与来自对方的声音一起传到听筒中, 而且自身的声音更强, 影响通话的质量。可将自身的声音作为噪声源 n_1 输入 ADALINE 中, 混有对方声音的信号作为目标响应, 通过抵消后, 系统就可以得到清晰的来自对方的声音信号。

这里, 假设传输信号为正弦波信号, 噪声为随机噪声, 进行自适应线性神经网络设计。



根据以上分析, ADALINE 自适应线性神经元的输入向量为随机噪声 n_t ; 正弦波信号与随机噪声之和为 ADALINE 神经元的目标向量; 输出信号为网络调整过程中的误差信号。

其实现的 MATLAB 代码为:

```
>>clear all;
%定义输入向量和目标向量
time=0.01:0.01:10;                %时间变量
noise=(rand(1,1000)-0.5)*4;        %随机噪声
input=sin(time);                   %信号
p=noise;                           %将噪声作为 ADALINE 的输入向量
t=input+noise;                     %将噪声+信号作为目标向量
%创建线性神经网络
net=newlin([-1 1],1,0,0.0005);
%线性神经网络的自适应调整(训练)
net.adaptParam.passes=70;
[net,y,output]=adapt(net,p,t);      %输出信号 output 为网络调整过程中的误差
%绘制信号, 叠加随机噪声的信号, 输出信号的波形
hold on;
%绘制信号的波形
subplot(3,1,1);plot(time,input,'r');
title('信号波形 sin(t)');
subplot(3,1,2);plot(time,t,'m');    %绘制叠加随机噪声信号的波形
xlabel('t');title('随机噪声波形 sin(t)+noise(t)');
%绘制输出信号的波形
subplot(3,1,3);plot(time,output,'b');
xlabel('t');title('输出信号波形 y(t)');
```

运行程序, 效果如图 3-12 所示。

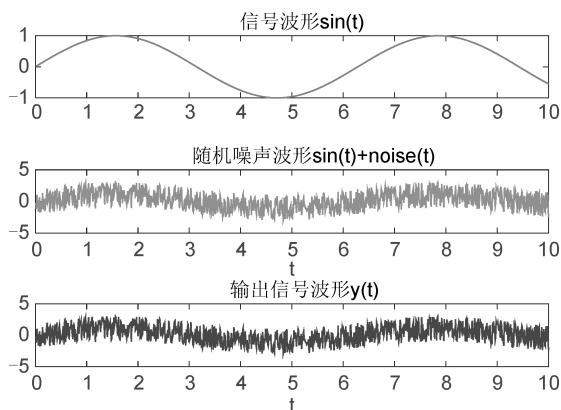


图 3-12 对消噪声的效果图

从图中可以看出, 输出信号除了含有一定直流分量外, 其波形与输入信号波形基本一致, 消除了叠加的随机噪声。



3.4 系统辨识

系统辨识是根据系统的输入输出时间函数来确定描述系统行为的数学模型。通过辨识建立数学模型的目的是估计表征系统行为的重要参数，建立一个能模仿真实系统行为的模型，用当前可测量的系统的输入和输出预测系统输出的未来演变并设计控制器。对系统进行分析的主要问题是根据输入时间函数和系统的特性来确定输出信号。对系统进行控制的主要问题是根据系统的特性设计控制输入，使输出满足预先规定的要求。而系统辨识所研究的问题恰好是这些问题的逆问题。通常，预先给定一个模型类 $\mu=\{M\}$ （即给定一类已知结构的模型），一类输入信号 u 和等价准则 $J=L(y, yM)$ （一般情况下， J 是误差函数，是过程输出 y 和模型输出 yM 的一个泛函）；然后选择使误差函数 J 达到最小的模型作为辨识所要求的结果。系统辨识包括两个方面：结构辨识和参数估计。线性网络可以用于对实际系统建模。如果实际系统是线性的或者接近线性的，则线性系统辨识误差会很小。

下面通过示例来演示线性神经网络在系统辨识中的应用。

【例 3-6】 利用线性网络进行系统辨识。

```
>> clear all;
%定义输入信号
t=0:0.025:5;
X=sin(sin(t.*t*10));
%定义系统变换函数 y=kx+b，并绘制系统输入 X 与输出 T 曲线
T=2*X+0.75;
figure;
plot(t,X,t,T,'');
legend('X 曲线','T 曲线')
%将信号 T 延时 0~2 个时间步长，得到网络的输入 X1
Q=length(T);
X1=zeros(3,Q);
X1(1,1:Q)=T(1,1:Q);
X1(2,2:Q)=T(1,1:(Q-1));
X1(3,3:Q)=T(1,1:(Q-2));
net=newlind(X1,T);
%网络仿真
y=sim(net,X1);
%绘制网络预测输出 y 与系统输出 T 及其误差
figure;plot(t,y,'+',t,T,'.',t,T-y,'x');
legend('预测输出系统曲线','系统输出曲线','误差曲线')
```

运行程序，效果如图 3-13 及图 3-14 所示。

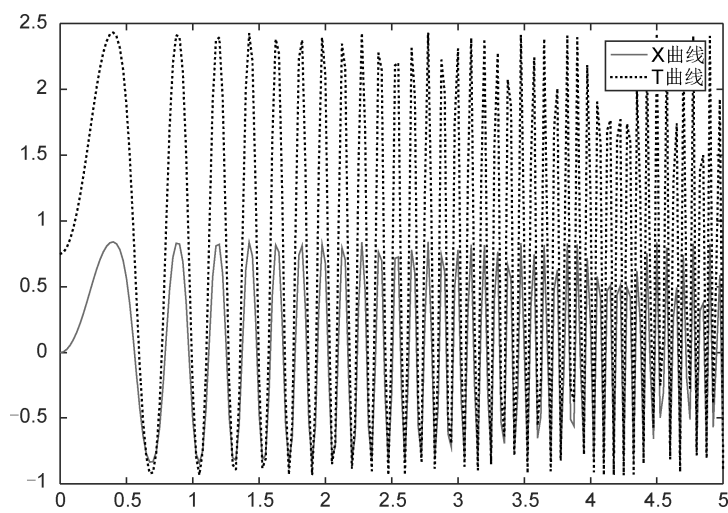


图 3-13 系统输入/输出曲线

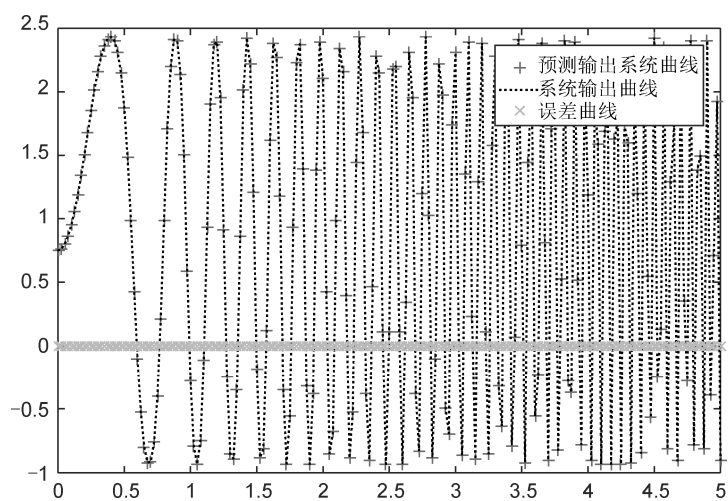


图 3-14 网络预测输出、系统输出及误差曲线

3.5 系统预测

实际物理系统的输入/输出关系式往往比较复杂，一般的形式为：

$$y(k) = f(x(k-n), x(k-n+1), \dots, x(k)) \quad (3-3)$$

由式 (3-3) 通过适当的设计网络的过程，使得式 (3-4) 成立：

$$a(k) = x(k) = f(x(k-1), \dots, x(k-n)) \quad (3-4)$$

就可以通过 $k-1$ 及以前测量的数据来完成 k 时刻的预测任务。设计结构如图 3-15 所示。

在这里需要特别注意， $x(k)$ 信号没有加在输入端。由此可见，训练网络的结构设计不是一成不变、生搬硬套的，一定要根据具体问题灵活应用和掌握。

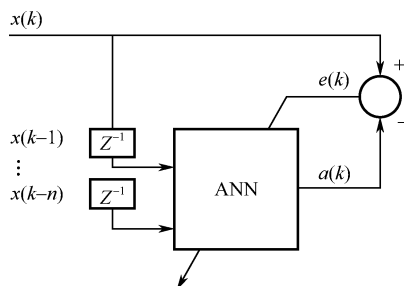


图 3-15 神经网络进行预测应用的设计结构图

设计的目的是使 $x(k)-a(k)=e(k)=0$ ，即当求得 $e=0$ 时，就可以得到网络输出 $a(k)=x(k)$ 的预测。预测的应用中同样有一个需要确定输出与前几次延时阶次有关的问题，只有 r （或 n ）确定正确了，预测的结果才准确。预测功能可以用在已知最近几年的产量或数据，预测明年的数值这类问题上。注意只有确认所要预测的关系式确实是具有线性关系时，方可采用自适应线性网络来实现。

由此可见，都是自适应线性元件网络，不同的网络设计结构能完成不同的任务，解决不同问题。灵活应用这种神经网络，就是要学会如何把不同的问题和网络结合起来，使之从数学理论上完成某种功能。虽然网络训练的都只是外部的输入/输出特性，但具体到每个应用时是有很大的不同的。

下面通过示例演示自适应预测的线性网络。

【例 3-7】 设计一个自适应线性网络，并对输入信号进行预测。输入为一线性调频信号，信号的采样时间为 2s，采样频率为 1000Hz，起始信号的瞬时为 0Hz，1s 时的瞬时频率为 150Hz。

```
>> clear all;
t=0:0.001:2;
time=0:0.001:2;
t=chirp(time,0,1,150); %产生线性调频信号
plot(time,t);
axis([0 0.5 -1 1]);
hold on;
xlabel('时间/s');ylabel('幅值');
T=con2seq([t]); %将矩阵转换为向量
P=T; %用延迟的信号作为样本的输入
title('信号预测');
lr=0.1; %神经网络的学习率
delays=[1 2 3 4 5]; %5 个延迟信号作为输入
net=newlin(minmax(cat(2,P{:})),1,delays,lr); %设计神经网络
[net,y,e]=adapt(net,P,T); %网络的自适应
plot(time,cat(2,y{:}),'r-','time,cat(2,T{:}),'g','time,cat(2,e{:}),'+'); %显示预测结果
legend('信号曲线','预测曲线','误差曲线')
```

运行程序，效果如图 3-16 所示。

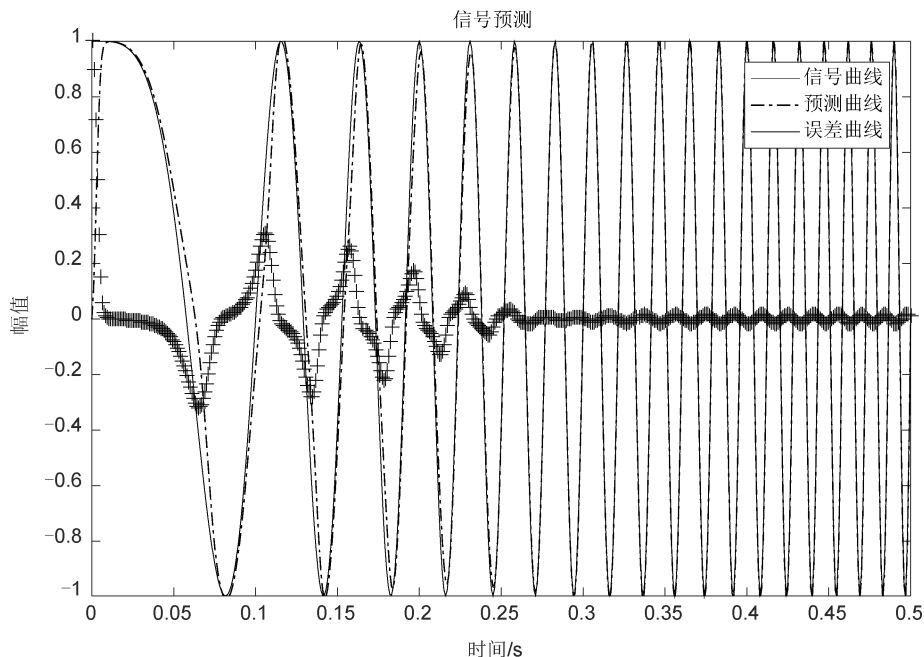
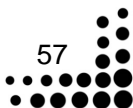


图 3-16 信号与网络预测及误差图

由图 3-16 中的误差曲线可见，在预测的初始阶段，误差较大，但经过一段时间（5 个信号）后，误差几乎趋于零，这是因为在初始阶段，网络的输入需 5 个延迟信号，输入不完整，因此不可避免出现初始误差。

【例 3-8】 利用线性网络预测一个时变信号序列。

```
>> clear all;
%分别定义两段时间，对应信号的不同频率时段
t1=0:0.05:4;
t2=4.05:0.024:6;
t=[t1 t2];
%获得待预测的目标信号，并绘制网络待预测的目标信号
T=[cos(t1*4*pi) cos(t2*8*pi)];
plot(t,T);
xlabel('时间/s');ylabel('幅值');
title('原始信号');
X=T;
ld=[1 2 3 4 5];
lr=0.1;
net=newlin(minmax(X),1,ld,lr); %新建一个线性层
[net,e,y]=adapt(net,X,T); %对网络进行自适应训练
%绘制网络预测输入 y、目标信号 T 及其误差信号 e
figure;
plot(t,y,'r',t,X,'x',t,e,'o');
```





```
xlabel('时间/s');ylabel('幅值');  
legend('预测曲线','目标信号曲线','误差曲线');
```

运行程序，效果如图 3-17 及图 3-18 所示。

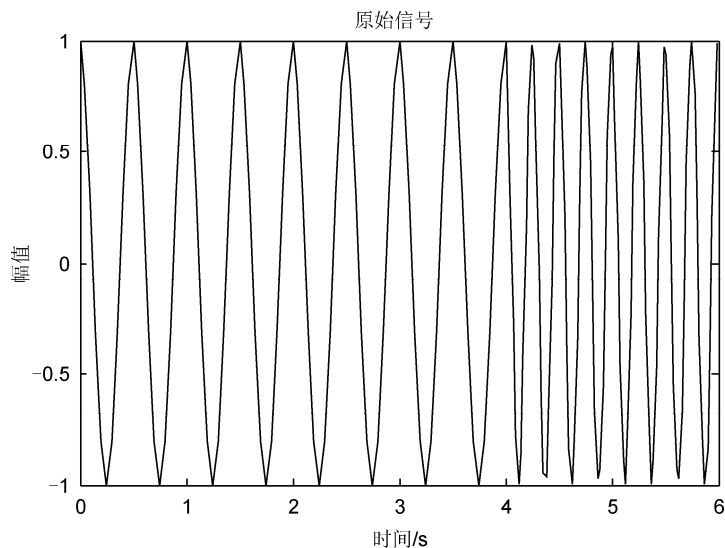


图 3-17 网络待预测的目标信号

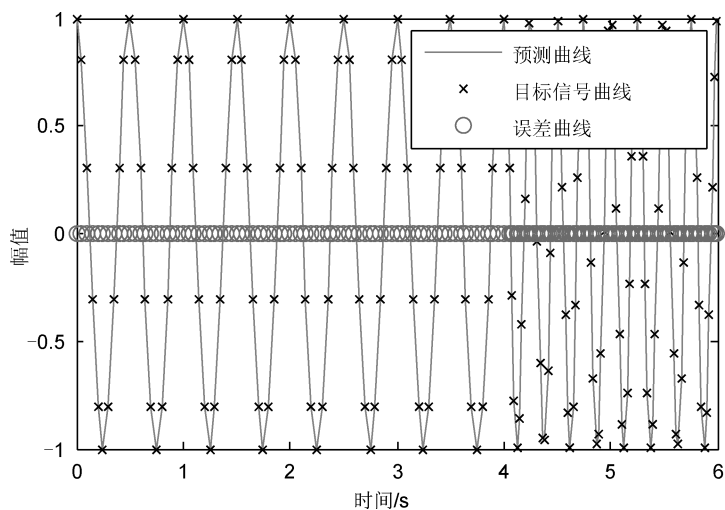


图 3-18 网络预测输出目标信号与误差

【例 3-9】 实现自适应预测的线性网络。设自适应滤波器如图 3-19 所示，该滤波器的目的是要从输入信号的前两个时刻的值预测当前时刻的值。

图中 D 为延迟单元，多个乞讨单元可以构成抽头延迟线，如图 3-20 所示。

设输入信号为一随机序列，试编写 MATLAB 程序，画出上述自适应滤波器的输入输出波形。

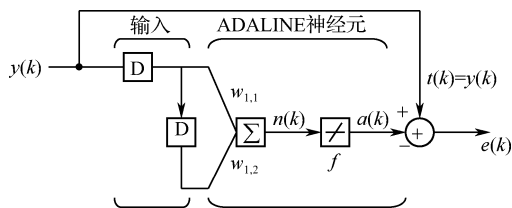


图 3-19 自适应滤波器示意图

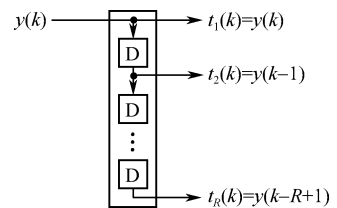


图 3-20 抽头延迟线

其实现的 MATLAB 程序代码如下：

```
>> clear all;
%定义输入向量和目标向量
time=0.5:0.5:20;                                %时间变量
y=(rand(1,40)-0.5)*4;                            %定义随机输入信号
p=con2seq(y);                                     %将随机输入向量转换为串行向量
delays=[1 2];                                     %定义 ADALINE 神经元输入延迟量
t=p;                                              %定义 ADALINE 神经元的数目向量
%创建线性神经网络
net=newlin(minmax(y),1,delays,0.0005);
%线性神经网络的自适应调整(训练)
net.adaptParam.passes=70;
[net,a,output]=adapt(net,p,t);                    %输出信号 output 为网络调整过程中的误差
%绘制随机输入信号\输出信号的波形
hold on;
subplot(3,1,1);plot(time,y,'r*-');              %输出信号 output 为网络调整过程中的误差
xlabel('t','position',[20.5,-1.8]);
ylabel('随机输入信号 s(t)');
axis([0 20 -2 2]);
subplot(3,1,2);
output=seq2con(output);
plot(time,output{1},'ko-');                       %绘制预测输出信号的波形
xlabel('t','position',[20.5,-1.8]);
ylabel('预测输出信号 y(t)');
axis([0 20 -2 2]);
subplot(3,1,3);
e=output{1}-y;
plot(time,e,'k-');                                %绘制误差曲线
xlabel('t','position',[20.5,-1.8]);
ylabel('误差曲线 e(t)');
axis([0 20 -2 2]);
hold off;
```

运行程序，效果如图 3-21 所示。

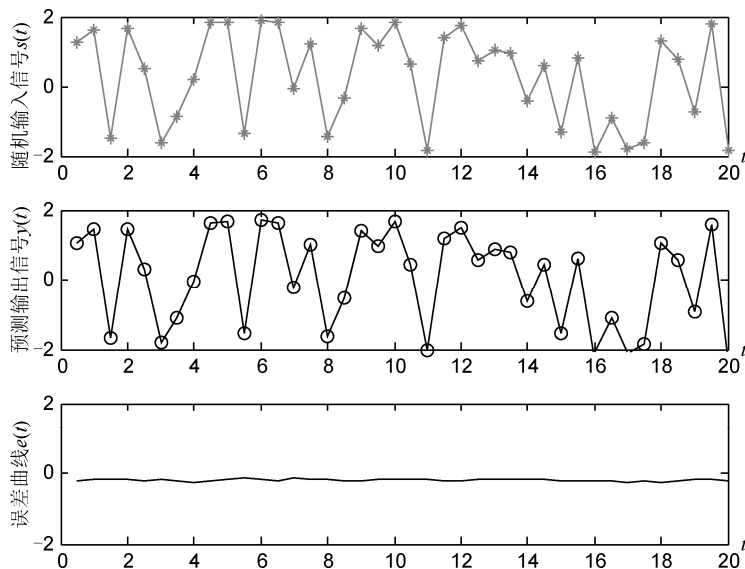


图 3-21 线性神经网络信号预测效果

从图中可以看出，输出信号波形与输入信号波形基本一致，误差较小，输出波形较好地预测了输入波形。

值得一提的是，在程序设计中，需要注意学习率和训练步长的选择。学习率过大，学习的过程将不稳定，且误差会更大；学习率过小，学习的过程将变慢，需要的训练步长数将加大。选择不当，将得不到满意的结果。

就线性神经网络本身而言，它与感知器一样，只能解决线性可分的模式分类，但 LMS 算法比感知器的 δ 学习算法更有效，因为它使均方误差最小，可以使各分类模式远离判决边界，从而使网络具有更好的抗噪性能。另一方面，ADALINE 网络至今仍然广泛应用于各种实际系统中，特别是在自适应滤波方面，用途更为广泛。

第 4 章 BP 网络算法分析与应用

BP 网络能学习和存储大量的输入-输出模式映射关系，而无须事前揭示描述这种映射关系的数学方程。它的学习规则是使用最速下降法，通过反向传播来不断调整网络的权值和阈值，使网络的误差平方和最小。BP 神经网络模型拓扑结构包括输入层、隐层和输出层。

4.1 BP 网络模型

BP 神经网络模型如图 4-1 所示。

BP 神经元与其他神经元类似，不同的是 BP 神经元的传输函数为非线性函数，最常用的函数是 logsig 和 tansig 函数，有的输出层也采用线性函数，其输出为：

$$a = \text{logsig}(Wp + b)$$

BP 网络一般为多层神经网络。由 BP 神经元构成的二层网络如图 4-2 所示。BP 网络的信息从输入层流向输出层，因此是一种多层前馈神经网络。

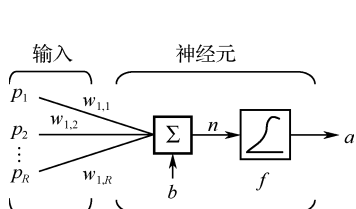


图 4-1 BP 神经元的一般模型

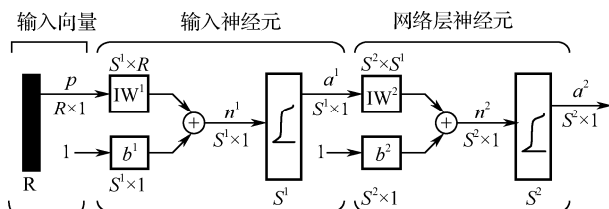


图 4-2 两层 BP 神经网络模型

如果多层 BP 网络的输出层采用 S 形传输函数（如 logsig ），其输出值将会限制在一个较小的范围内（0，1）；而采用线性传输函数则可以取任意值。

BP 神经网络模型 BP 网络模型包括输入输出模型、作用函数模型、误差计算模型和自学习模型。

1) 节点输出模型

隐节点输出模型为：

$$O_j = f(\sum W_{ij} X_i - q_j)$$

输出节点输出模型：

$$Y_k = f(\sum T_{jk} O_j - q_k)$$

其中， f 为非线性作用函数； q 为神经单元阈值。



2) 作用函数模型

作用函数是反映下层输入对上层节点刺激脉冲强度的函数, 又称刺激函数, 一般取 (0,1) 内连续取值 Sigmoid 函数:

$$f(x) = \frac{1}{1+e^{-x}}$$

3) 误差计算模型

误差计算模型是反映神经网络期望输出与计算输出之间误差大小的函数:

$$E_q = \frac{1}{2} \sum (t_{pi} - O_{pi})^2$$

其中, t_{pi} 为节点的期望输出值; O_{pi} 为节点计算输出值。

4) 自学习模型

神经网络的学习过程, 即连接下层节点和上层节点之间的权重矩阵 W_{ij} 的设定和误差修正过程。BP 网络分为有师学习方式-需要设定期望值和无师学习方式-只需输入模式。自学习模型为:

$$\Delta W_{ij}(n+1) = h\phi_i O_j + a\Delta W_{ij}(n)$$

其中, h 为学习因子; ϕ_i 为输出节点 i 的计算误差; O_j 为输出节点 j 的计算输出; a 为动量因子。

4.2 BP 网络学习算法

BP 网络的学习过程分为两个阶段:

第一个阶段是输入已知学习样本, 通过设置的网络结构和前一次迭代的权值和阈值, 从网络第一层向后计算各神经元的输出。

第二个阶段是对权值和阈值进行修改, 从最后一层向前计算各权值和阈值对总误差的影响 (梯度), 据此对各权值和阈值进行修改。

以上两个过程反复交替, 直到达到收敛为止。由于误差逐层往回传递, 以修正层与层之间的权值和阈值, 所以称该算法为误差反向传播算法, 这种误差反向传播学习算法可以推广到有若干个中间层的多层网络, 因此该多层网络常称之为 BP 网络。标准的 BP 算法和 Widrow-Hoff 学习规则一样是一种梯度下降学习算法, 其权值的修正是沿着误差性能函数梯度的反方向进行的。针对标准 BP 算法存在的一些不足, 出现了几种基于标准 BP 算法的改进算法, 如变梯度算法、牛顿算法等。

4.2.1 BP 网络学习算法介绍

1) 最速下降 BP 算法 (Steepest Descent Back Propagation, SDBP)

对于 BP 神经网络, 设 k 为迭代次数, 则每一层权值和阈值的修正按式 (4-1) 进行:



$$x(k+1) = x(k) - \alpha g(k) \quad (4-1)$$

式中, $x(k)$ 为第 k 次迭代各层之间的连接权向量或阈值向量。

$g(k) = \frac{\partial E(k)}{\partial x(k)}$ 为第 k 次迭代的神经网络输出误差对各权值或阈值的梯度向量。负号表示梯度的反方向, 即梯度的最速下降方向。

α 为学习速率, 在训练时是一常数。在 MATLAB 神经网络工具箱中, 其默认值为 0.01, 可以通过改变训练参数进行设置。

$E(k)$ 为第 k 次迭代网络输出的总误差性能函数, 在 MATLAB 神经网络工具箱中, BP 网络误差性能函数默认值为均方误差 MSE (Mean Square Error), 以二层 BP 网络为例, 只有一个输入样本时, 有

$$E(k) = E[e^2(k)] \approx \frac{1}{S^2} \sum_{i=1}^{S^2} [t_i^2 - a_i^2(k)]^2 \quad (4-2)$$

$$\begin{aligned} a_i^2(k) &= f^2 \left\{ \sum_{j=1}^{S^2} [w_{i,j}^2(k) a_i^1(k) - b_i^2(k)] \right\} \\ &= f^2 \left\{ \sum_{j=1}^{S^2} \left[w_{i,j}^2(k) f^1 \left(\sum_{j=1}^{S^1} (i w_{i,j}^1(k) p_i + i b_i^1(k)) \right) + b_i^2(k) \right] \right\} \end{aligned} \quad (4-3)$$

若有 n 个输入样本:

$$E(k) = E[e^2(k)] \approx \frac{1}{n S^2} \sum_{j=1}^{S^1} \sum_{i=1}^{S^2} [t_i^2 - a_i^2(k)]^2 \quad (4-4)$$

根据式 (4-2) 或式 (4-4) 和各层的传输函数, 可以求出第 k 次迭代的总误差曲面的梯度 $g(k) = \frac{\partial E(k)}{\partial x(k)}$, 分别代入式 (4-1) 便可以逐次修正其权值和阈值, 并使总的误差向减小的方向变化, 直到达到所要求的误差性能为止。

从上述过程可以看出, 权值和阈值的修正是在所有样本输入后, 计算其总的误差后进行的, 这种修正方式称为批处理。在样本数比较多时, 批处理方式比分别处理方式的收敛速度快。

2) 动量 BP 算法 (Momentum Backpropagation, MOBP)

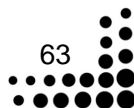
动量 BP 算法是在梯度下降算法的基础上引入动量因子 $\eta (0 < \eta < 1)$:

$$\Delta x(k+1) = \eta \Delta x(k) + \alpha (1 - \eta) \frac{\partial E(k)}{\partial x(k)} \quad (4-5)$$

$$x(k+1) = x(k) + \Delta x(k+1) \quad (4-6)$$

该算法是以前一次的修正结果来影响本次修正量, 当前一次的修正量过大时, 式 (4-5) 第二项的符号将与前一次修正量的符号相反, 从而使本次的修正量减小, 起到减小振荡的作用; 当前一次的修正量过小时, 式 (4-5) 第二项的符号将与前一次修正量的符号相同, 从而使本次的修正量增大, 起到加速修正的作用。可以看出, 动量 BP 算法总是力图使在同一梯度方向上的修正量增加。动量因子 η 越大, 同一梯度方向上的动量也越大。

在动量 BP 算法中, 可以采用较大的学习率, 而不会造成学习过程的发散, 因为当修正过





量时,该算法(动量 BP 算法)总是可以使修正量减小,以保持修正方向向着收敛的方向进行;另一方面,动量 BP 算法总是加速同一梯度方向的修正量。上述两个方面表明,在保证算法稳定的同时,动量 BP 算法的收敛速率较快,学习时间较短。

3) 学习率可变的 BP 算法 (Variable Learning rate Backpropagation, VLBP)

在最速下降 BP 算法和动量 BP 算法中,其学习率是一个常数,在整个训练的过程中保持不变,学习算法的性能对于学习率的选择非常敏感,学习率过大,算法可能振荡而不稳定;学习率过小,则收敛速度慢,训练的时间长。而在训练之前,要选择最佳的学习率是不现实的。事实上,可以在训练的过程中使学习率随之变化,而使算法沿着误差性能曲面进行修正案。

自适应调整学习率的梯度下降算法,在训练的过程中,力图使算法稳定,而同时又使学习的步长尽量大,学习率则是根据局部误差曲面作出相应的调整。当误差以减小的方式趋于目标时,说明修正方向正确,可使步长增加,因此学习率乘以增量因子 k_{inc} ,使学习率增加;而当误差增加超过事先设定值时,说明修正过头,应减小步长,因此学习率乘以减量因子 k_{dec} ,使学习率减小,同时舍去使误差增加的前一步修正过程,即

$$a(k+1) = \begin{cases} k_{inc}a(k) & E(k+1) < E(K) \\ k_{dec}a(k) & E(k+1) > E(K) \end{cases} \quad (4-7)$$

4) 弹性算法 (Resilient back-PROPagation, RPROP)

多层 BP 网络的隐层一般采用传输函数 sigmoid,它把一个取值范围为无穷大的输入变量压缩到一个取值范围有限的输出变量中。函数 sigmoid 具有这样的特性:当输入变量的取值很大时,其斜率趋于零,这样在采用最速下降 BP 法训练传输函数为 sigmoid 的多层网络时就带来一个问题,尽管权值和阈值离其最佳值相差甚远,但此时梯度的幅度非常小,导致权值和阈值的修正量也很小,这样就使训练的时间变得很长。

RPROP 算法的目的是消除梯度幅度的不利影响,所以在进行权值的修正时,仅仅用到偏导的符号,而其幅值却不影响权值的修正,权值大小的改变取决于与幅值无关的修正值。当连续两次迭代的梯度方向相同时,可将权值和阈值的修正值乘以一个增量因子,使其修正值增加;当连续两次迭代的梯度方向相反时,可将权值和阈值的修正值乘以一个减量因子,使其修正值减小;当梯度为零时,权值和阈值的修正值保持不变;当权值的修正发生振荡时,其修正值将会减小。如果权值在相同的梯度上连续被修正,则其幅度必将增加,从而克服了梯度幅度偏导的不利影响,即

$$\begin{aligned} \Delta x(k+1) &= \Delta x(k) \text{sign}(g(k)) \\ &= \begin{cases} \Delta x(k)k_{inc} \text{sign}(g(k)) & (\text{当连续两次迭代的梯度方向相同时}) \\ \Delta x(k)k_{dec} \text{sign}(g(k)) & (\text{当连续两次迭代的梯度方向相反时}) \\ \Delta x(k) & (\text{当 } g(k) = 0 \text{ 时}) \end{cases} \end{aligned} \quad (4-8)$$

式中, $g(k)$ 为第 k 次迭代的梯度; $\Delta x(k)$ 为权值或阈值第 k 次迭代的幅度修正值,其初始值 $\Delta x(0)$ 是用户设置的;增量因子 k_{inc} 和减量因子 k_{dec} 也是用户设置的。



5) 变梯度算法 (Conjugate Gradient Backpropagation, CGBP)

最速下降 BP 算法是沿着梯度最陡下降方向修正权值的, 虽然误差函数沿着梯度的最陡下降方向进行修正, 误差减小的速度是最快的, 但收敛的速度不一定是最快的。在变梯度算法中, 沿着变化的方向进行搜索, 使其收敛速度比最陡下降梯度方向的收敛速度更快。

所有变梯度算法的第一次迭代都是沿着最陡梯度下降方向开始进行搜索的:

$$p(0) = -g(0) \quad (4-9)$$

然后, 决定最佳距离的线性搜索沿着当前搜索的方向进行:

$$x(k+1) = x(k) + \alpha p(k) \quad (4-10)$$

$$p(k) = -g(k) + \beta(k)p(k-1) \quad (4-11)$$

式中, $p(k)$ 为第 $k+1$ 次迭代的搜索方向, 从式 (4-11) 可以看出, 它由第 k 次迭代的梯度和搜索方向共同决定; 系数 $\beta(k)$ 在不同的变梯度设法中有不同的计算方法。

● Fletcher-Reeves 修正算法

Fletcher-Reeves 修正算法是由 R.Fletcher 和 C.M.Reeves 提出的, 在式 (4-11) 中, 系数 $\beta(k)$ 定义为:

$$\beta(k) = \frac{g^T(k)g(k)}{g^T(k-1)g(k-1)} \quad (4-12)$$

这种变梯度算法的速度通常比变学习率算法的速率快得多, 有时比 RPROP 算法还快。其所需的存储空间也比普通算法略多一点, 所以在连接权的数量很多时, 时常选用该算法。

● Polak-Ribiere 修正算法

Polak-Ribiere 算法是由 Polak 和 Ribiere 提出的, 在式 (4-11) 中, 系数 $\beta(k)$ 定义为:

$$\beta(k) = \frac{\Delta g^T(k-1)g(k)}{g^T(k-1)g(k-1)} \quad (4-13)$$

此时 Fletcher-Reeves 修正算法就演变成 Polak-Ribiere 修正算法。

Polak-Ribiere 修正算法的性能与 Fletcher-Reeves 修正算法相差无几, 但存储空间比 Fletcher-Reeves 修正算法略大。

● Powell-Beale 复位算法

对于所有的变梯度算法, 搜索方向都会周期性地被复位成负的梯度方向, 通常复位点出现在迭代次数和网络参数个数 (权值和阈值) 相等的地方, 为了提高训练的有效性, 另外一些复位的算法被提出, 其中 Powell-Beale 复位算法是 Beale 和 Powell 首先提出的。在此算法中, 如果梯度满足下式:

$$|g^T(k-1)g(k)| \gg 0.2 \|g(k)\|^2 \quad (4-14)$$

则搜索方向被复位成负的梯度方向, 即 $p(k) = -g(k)$ 。

尽管对于任意给定的一个问题, 该算法的性能难以预先确定, 但可以肯定, 在处理某些问题上 Powell-Beale 复位算法的性能比 Polak-Ribiere 修正算法要略好些, 其存储空间则比 Polak-Ribiere 修正算法要略大些。

● SCG (Scaled Conjugate Gradient) 算法

到目前为止我们讨论的各种变梯度算法在每次迭代时都需要确定线性搜索方向, 而线性



搜索的计算需要付出的代价是很大的, 因为每一次搜索都需要对全部训练样本的网络响应进行多次计算。SCG 算法是由 Moller 提出的改进算法, 它不需要在每一次迭代中都进行线性搜索, 从而避免了搜索方向计算的耗时问题, 其基本思想采用了模型信任区间逼近的原理。

SCG 算法也许比其他变梯度算法需要更多的迭代次数, 但由于不需要在迭代中进行线性搜索, 所以每次迭代的计算量大大减小。SCG 算法所需要的存储空间与 Fletcher-Reeves 修正算法的存储空间相差无几。

6) 拟牛顿算法 (Quasi-Newton Algorithms)

牛顿法是一种基于二阶泰勒 (Taylor) 级数的快速优化算法, 其基本方法是:

$$x(k+1) = x(k) - A^{-1}(k)g(k) \quad (4-15)$$

式中, $A(k)$ 为误差性能函数在当前权值和阈值下的 Hessian 矩阵 (二阶导数):

$$A(k) = \nabla^2 F(x)|_{x=x(k)} \quad (4-16)$$

牛顿法通常比变梯度法的收敛速度快, 但对于前馈神经网络计算 Hessian 矩阵是很复杂的, 付出的代价也很大。

有一类基于牛顿法的算法不要求二阶导数, 此类方法称为拟牛顿法 (或正切法), 在算法中的 Hessian 矩阵用其近似值进行修正, 修正值被看成梯度的函数。

● BFGS (Broyden, Fletcher, Goldfarb and Shanno) 算法

在公开发表的研究成果中, 拟牛顿法应用最为成功的有 Broyden, Fletcher, Goldfarb 和 Shanno 修正算法, 合称为 BFGS 算法。

该算法虽然收敛所需的步长通常较少, 但在每次迭代过程中所需要的计算量和存储空间比变梯度算法都要大, 对近似 Hessian 矩阵必须进行存储, 其大小为 $n \times n$, 这里 n 为网络的连接权和阈值的数量。所以对于规模很大的网络用 RPROP 算法或任何一种变梯度算法可能好些, 而对于规模较小的网络则用 BFGS 算法可能更有效。

● OSS (One Step Secant) 算法

由于 BFGS 算法在每次迭代时比变梯度算法需要更多的存储空间和计算量, 所以对于正切近似法减少其存储量和计算量是必要的。OSS 算法试图解决变梯度法和拟牛顿 (正切) 法之间的矛盾, 该算法不必存储全部 Hessian 矩阵, 它假定每一次迭代时前一次迭代的 Hessian 矩阵具有一致性, 这样做的另外一个优点是, 在新的搜索方向进行计算时不必计算矩阵的逆。

该算法每次迭代所需的存储量和计算量介于梯度算法和完全拟牛顿算法之间。

7) LM (Levenberg-Marquardt) 算法

LM 算法与拟牛顿法一样, 是为了在以近似二阶训练速率进行修正时避免计算 Hessian 矩阵而设计的。当误差性能函数具有平方和误差 (训练前馈网络的典型误差函数) 的形式时, Hessian 矩阵可以近似表示为:

$$H = J^T J \quad (4-17)$$

梯度的计算表达式为:

$$g = J^T e \quad (4-18)$$

式中, H 是包含网络误差函数对权值和阈值一阶导数的雅可比矩阵, e 是网络的误差向量。雅可比矩阵可以通过标准的前馈网络技术进行计算, 比 Hessian 矩阵的计算要简单得多。



类似于牛顿法, LM 算法用上述近似 Hessian 矩阵按式 (4-18) 进行修正。

$$x(k+1) = x(k) - [J^T J + \mu J]^{-1} J^T e \quad (4-18)$$

当系数 μ 为 0 时, 式 (4-18) 即为牛顿法; 当系数 μ 的值很大时, 式 (4-18) 变为步长较小的梯度下降法。牛顿法逼近最小误差的速度更快、更精确, 因此应尽可能使算法接近于牛顿法, 在每一步成功的迭代后 (误差性能减小), 使 μ 减小; 仅在进行尝试性迭代后的误差性能增加的情况下, 才使 μ 增加。这样, 该算法每一步迭代的误差性能总是减小的。

LM 算法是为了训练中等规模的前馈神经网络 (多达数百个连接权) 而提出的最快速算法, 它对 MATLAB 实现也是相当有效的, 因为其矩阵的计算在 MATLAB 中是以函数实现的, 其属性在设置时变得非常明确。

4.2.2 BP 网络学习算法的比较

对于一个给定的问题, 到底采用哪种训练方式其训练速度是最快, 这里很难预知的, 因为这取决于许多因素, 包括给定问题的复杂性、训练样本集的数量、网络权值和阈值的数量、误差目标、网络的用途 (如用于模式识别还是函数逼近) 等。

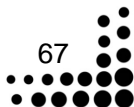
但通过实验, 可以得出各种算法性能上的一些结论, 通常对于包含数百个权值的函数逼近网络, LM 算法的收敛速度最快。如果要求的精度比较高, 则该算法的优点尤其突出。在许多情况下, 采用 LM 算法的训练函数 `trainlm` 可以获得比其他任何一种算法更小的均方误差。但当网络权值的数量增加时, `trainlm` 的优点将逐渐变得不很明显。另外, `trainlm` 对于模式识别相关问题的处理功能很弱, 其存储空间比其他算法大, 通过调整 `trainlm` 的存储空间参数 `mem_reduc`, 虽然可以在一定程度上减小对存储空间的要求, 但却需要增加运行时间。

将 RPROP 算法的训练函数 `trainrp` 应用于模式识别时, 其速度是最快的, 但对于函数逼近问题该算法却不是最好的, 其性能同样会随着目标误差的减小而变差。该算法所需的存储空间较其他算法相对要小一些。

变梯度算法, 特别是 SCG 算法, 在更广泛的问题中, 尤其在网络规模较大的场合, 其性能都很好。SCG 算法应用于函数逼近问题时, 几乎与 LM 算法一样快 (在网络规模较大时比 LM 算法更快); 而应用于模式识别时几乎与 RPROP 算法一样快, 其性能不像 RPROP 算法随着目标误差的减小而下降得那么快。变梯度算法对存储空间的要求相对也低一些。

BFGS 算法类似于 LM 算法, 其所需的存储空间比 LM 算法小, 但其运算量却随网络的大小成几何级数增长, 因为对每次迭代过程都必须计算相应矩阵的逆矩阵。

变学习率算法通常比其他算法的速度要慢很多, 而其存储空间与 RPROP 算法一样, 但在应用于某些问题时该算法仍然很有用。在有些特定的情形下收敛速度慢一些反而好, 例如, 如果用收敛速度太快的算法, 可能得到的结果还达不到所要求的目标时训练就提前结束了, 会错过使误差最小的点。





4.3 BP 神经网络特点

BP 神经网络的信息处理方式具有如下特点:

1) 信息分布存储

人脑存储信息的特点是利用突触效能的变化来调整存储内容,即信息存储在神经元之间连接强度的分布上,BP 神经网络模拟人脑的这一特点,使信息以连接权值的形式分布于整个网络。

2) 信息并行处理

人脑神经元之间传递脉冲信号的速度远低于冯·诺依曼计算机的工作速度,但是在很多问题上却可以快速判断、决策和处理,这是由于人脑是一个大规模并行与串行组合的处理系统。BP 神经网络的基本结构模仿人脑,具有并行处理的特征,大大提高了网络功能。

3) 具有容错性

生物神经系统部分不严重损伤并不影响整体功能,BP 神经网络也具有这种特性,网络的高度连接意味着少量的误差可能不会产生严重的后果,部分神经元的损伤不破坏整体,它可以自动修正误差。这与现代计算机的脆弱性形成鲜明对比。

4) 具有自学习、自组织、自适应的能力

BP 神经网络具有初步的自适应与自组织能力,在学习或训练中改变突触权值以适应环境,可以在使用过程中不断学习完善自己的功能,并且同一网络因学习方式的不同可以具有不同的功能,它甚至具有创新能力,可以发展知识,甚至超过设计者原有的知识水平。

4.4 BP 网络功能

目前,在人工神经网络的实际应用中,绝大部分的神经网络模型都采用 BP 神经网络及其变化形式。它也是前向网络的核心部分,体现了人工神经网络的精华。

BP 网络主要用于以下 4 方面。

- (1) 函数逼近:用输入向量和相应的输出向量训练一个网络以逼近一个函数。
- (2) 模式识别:用一个待定的输出向量将它与输入向量联系起来。
- (3) 分类:把输入向量所定义的合适方式进行分类。
- (4) 数据压缩:减少输出向量维数以便传输或存储。

4.5 BP 网络实例分析

下面通过举几个例子来说明 BP 网络的应用。



1) 应用学习算法来分析 BP 网络

【例 4-1】 采用动量梯度下降算法训练 BP 网络。

```
>> clear all
echo on
%newff 生成一个新的前向神经网络
%train 对 BP 神经网络进行训练
%Ssim 对 BP 神经网络进行仿真
%定义训练样本
P=[-1, -2, 3, 1; -1, 1, 5, -3]; %P 为输入向量
T=[-1, -1, 1, 1]; %T 为目标向量
%创建一个新的前向神经网络
net=newff(minmax(P),[3,1],{'tansig','purelin'},'traingdm');
%当前输入层权值和阈值
inputWeights=net.IW{1,1}
inputbias=net.b{1}
%当前网络层权值和阈值
layerWeights=net.LW{2,1}
layerbias=net.b{2}
%设置训练参数
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.epochs = 1000;
net.trainParam.goal = 1e-3;
%调用 traingdm 算法训练 BP 网络
[net,tr]=train(net,P,T);
%对 BP 网络进行仿真
A = sim(net,P)
%计算仿真误差
E = T-A
MSE=mse(E)
echo off
```

运行程序，BP 网络训练过程图如图 4-3 所示。

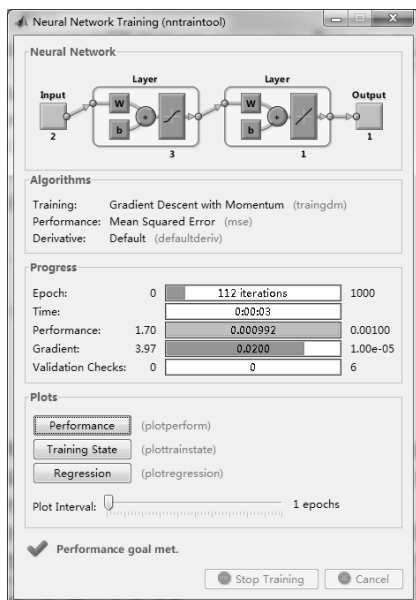


图 4-3 BP 网络训练过程

运行程序，输出如下：

```
inputWeights =
    -0.4496    0.5372
     0.3057   -0.5753
    -0.8027   -0.3403
inputbias =
     2.1125
     0.4225
    -1.6832
layerWeights =
    -0.9077   -0.8057    0.6469
layerbias =
     0.3897
A =
    -0.9534   -1.0406    1.0105    0.9937
E =
    -0.0466    0.0406   -0.0105    0.0063
MSE = 9.9164e-04
```

2) 用于模式识别与分类的 BP 网络

【例 4-2】 以 BP 神经网络实现对图 4-4 所示两类模式的分类。

由图 4-4 所示两类模式可以看出，分类为简单的非线性分类。有一个输入向量，包含两个输入元素；两类模式，一个输出元素即可表示；可以以图 4-5 所示两层 BP 网络来实现分类。

根据图 4-4 所示两类模式确定的训练样本为：



$$p = \begin{bmatrix} 1 & -1 & -2 & -4 \\ 2 & 1 & 1 & 0 \end{bmatrix}, \quad t = [0.2 \quad 0.8 \quad 0.8 \quad 0.2]$$

因为 BP 网络的输出为 **logsig** 函数，所以目标向量的取值为 0.2 和 0.8，分别对应两类模式。在程序设计时，通过判决门限 0.5 区分两类模式。

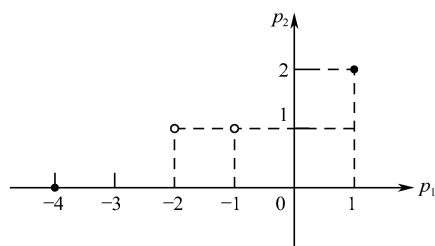


图 4-4 待分类模式

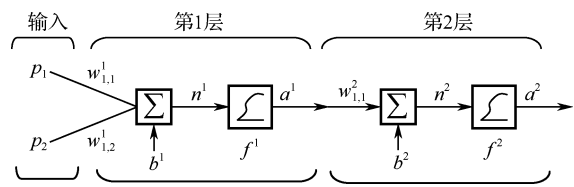


图 4-5 两层 BP 网络

因为处理的问题简单，所以采用最速下降 BP 算法（**traingd** 训练函数）训练该网络，以熟悉该算法的应用。

```
%创建和训练 BP 的 MATLAB 程序
clear all
%定义输入向量和目标向量
P=[1 2;-1 1; -2 1; -4 0]';
T=[0.2 0.8 0.8 0.2];
%创建 BP 网络和定义训练函数及参数
net=newff([-1 1; -1 1],[5 1],{'logsig' 'logsig'},'traingd');
net.trainParam.goal=0.001;
net.trainParam.epochs=5000;
%训练神经网络
[net,tr]=train(net,P,T);
%输出训练后的权值和阈值
iw1=net.iw{1}
b1=net.b{1}
iw2=net.lw{2}
b2=net.b{2}
```

BP 网络的初始化函数的默认值为 **initnw**，得到训练过程如图 4-6 所示。

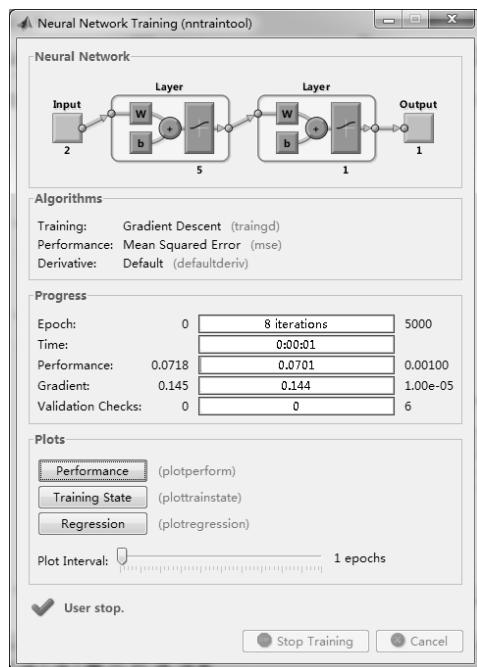


图 4-6 例 4-1 的训练过程图

得到训练结果如下：

```
iw1 =
    -5.7464    2.4873
    -1.2074   -5.9623
     4.9746    3.8462
     3.5326    5.1814
     6.0123    2.0854
b1 =
     6.2561
     3.3419
    -0.1803
     3.1184
     6.0767
iw2 =
     2.8061     2.4575    -1.3763     1.9674    -3.2571
b2 =
    -1.1616
```

由图 4-6 可以看出，训练经过了 5000 次仍然未达到要求的目标误差 0.001，说明采用训练函数 traingd 进行训练的收敛速度是很慢的。

虽然训练的误差性能未达到要求的目标误差，但这并不妨碍我们以测试样本对网络进行仿真。



```
%网络仿真的 MATLAB 程序
p=[1 2; -1 1; -2 1; -4 0]';
a2=sim(net,p1)
b2=a2>0.5
```

得到仿真结果为:

```
a2 =
    0.2625    0.7040    0.9852    0.9837
b2 =
     0         1         1         1
```

3) 用于曲线拟合的 BP 网络

在实际应用中, 往往希望产生一些非线性的输入输出曲线, 且没有明确的函数关系, 借助神经网络实现曲线拟合, 可以很方便地解决这一问题。

【例 4-3】 已知某系统输出 y 与输入 x 部分的对应关系如表 4-1 所示。设计一 BP 神经网络, 完成 $y = f(x)$ 。

表 4-1 函数 $y=f(x)$ 的对应关系

x	-1	-0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1
y	-0.832	-0.423	-0.024	0.344	1.282	3.456	4.02	3.232	2.102	1.504
x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
y	0.248	1.242	2.344	3.262	2.052	1.684	1.022	2.224	3.022	1.984

以隐层节点数为 15 的单输入和单输出两层 BP 网络来实现曲线拟合。
创建和训练 BP 网络的 MATLAB 程序。

```
clear all;
P=-1:0.1:0.9;
T=[-0.832 -0.423 0.024 0.344 1.282 3.456 4.02 3.232 2.102 1.504...
    0.248 1.242 2.344 3.262 2.052 1.684 1.022 2.224 3.022 1.984];
net=newff([-1 1],[15 1],{'tansig' 'purelin'},'traingdx','learnqdm');
net.trainParam.epochs=2500;
net.trainParam.goal=0.001;
net.trainParam.show=10;
net.trainParam.lr=0.05;
net=train(net,P,T);
```

训练过程如图 4-7 所示。

由图 4-7 可看出, 当训练次数为 186 时, 其训练误差为 $0.000\ 978 < 0.001$, 即满足要求。

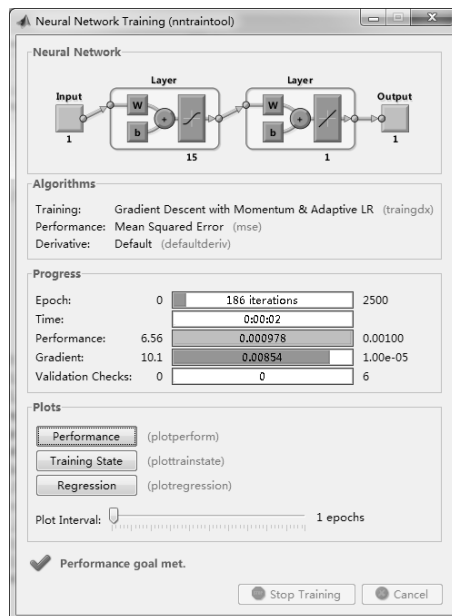


图 4-7 训练过程

BP 网络实现的拟合程序为:

```
>> P=-1:0.1:0.9;
T=[-0.832 -0.423 0.024 0.344 1.282 3.456 4.02 3.232 2.102 1.504...
    0.248 1.242 2.344 3.262 2.052 1.684 1.022 2.224 3.022 1.984];
hold on
plot(P,T,'r+');
p=-1:0.01:0.9;
R=sim(net,p);
plot(p,R);
hold off
```

运行程序，得到曲线拟合效果如图 4-8 所示。

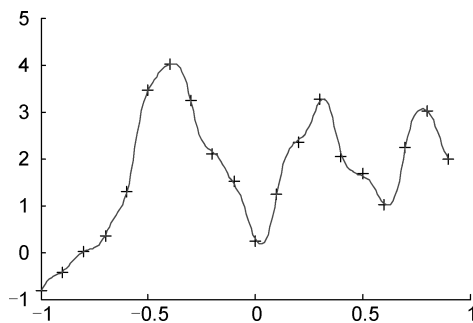


图 4-8 BP 网络拟合效果图

实线为得到的拟合曲线；“+”为训练样本。从结果上看，可以对个别训练样本进行很好



的拟合，但拟合曲线欠光滑，出现了“过适配”现象。如果改用 `trainbr` 训练函数进行训练，则曲线拟合会变得更光滑一些，在此希望读者自行动手试一试其效果图。

【例 4-4】 采用贝叶斯正则化算法提高 BP 网络的推广能力。在本例中，采用两种训练方法，即 L-M 优化算法 (`trainlm`) 和贝叶斯正则化算法 (`trainbr`)，用于训练 BP 网络，使其能够拟合某一附加有白噪声的正弦样本数据。其中，样本数据可以采用如下 MATLAB 语句生成：

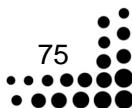
输入向量： `P = [-1:0.05:1];`

目标向量： `randn('seed',78341223);`

`T = sin(2*pi*P)+0.1*randn(size(P));`

实现的 MATLAB 程序如下：

```
>> clear all
%定义训练样本向量
P = [-1:0.05:1]; %P 为输入向量
%T 为目标向量
randn('seed',78341223); T = sin(2*pi*P)+0.1*randn(size(P));
%绘制样本数据点
plot(P,T,'+');
hold on;
plot(P,sin(2*pi*P),'-'); %绘制不含噪声的正弦曲线
%创建一个新的前向神经网络
net=newff(minmax(P),[20,1],{'tansig','purelin'});
disp('1. L-M 优化算法 TRAINLM'); disp('2. 贝叶斯正则化算法 TRAINBR');
choice=input('请选择训练算法(1,2):');
figure(gcf);
if(choice==1)
    %采用 L-M 优化算法 TRAINLM
    net.trainFcn='trainlm';
    %设置训练参数
    net.trainParam.epochs = 500;
    net.trainParam.goal = 1e-6;
    net=init(net);
    %重新初始化
elseif(choice==2)
    %采用贝叶斯正则化算法 TRAINBR
    net.trainFcn='trainbr';
    %设置训练参数
    net.trainParam.epochs = 500;
    randn('seed',192736547);
    net = init(net);
    %重新初始化
end
%调用相应算法训练 BP 网络
```



```
[net,tr]=train(net,P,T);
%对 BP 网络进行仿真
A=sim(net,P);
%计算仿真误差
E=T-A;
MSE=mse(E)
%绘制匹配结果曲线
plot(P,A,P,T,'+',P,sin(2*pi*P),'-');
```

运行结果输出如下：

1. L-M 优化算法 TRAINLM。
2. 贝叶斯正则化算法 TRAINBR。

请选择训练算法 (1,2):

当在命令窗口中输入“1”时，得到利用 L-M 优化算法 TRAINLM 的训练过程图 (图 4-9)、拟合效果图 (图 4-10) 及运行结果。

MSE =
9.8274e-07

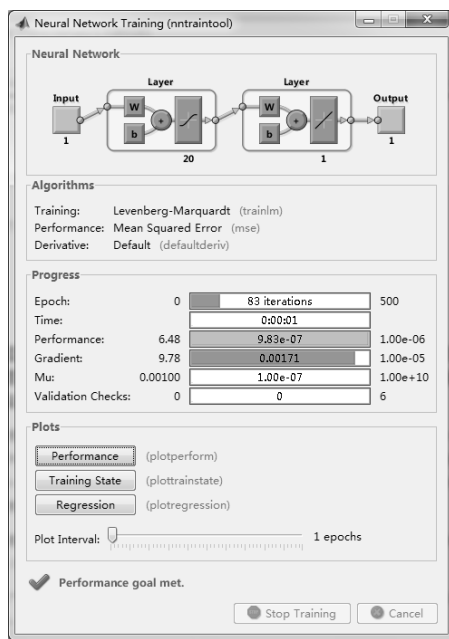


图 4-9 L-M 优化算法训练过程图

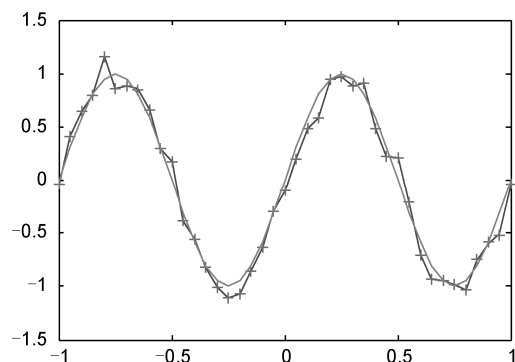


图 4-10 L-M 优化算法拟合效果图

当在命令窗口中输入“2”时，得到利用贝叶斯正则化算法 TRAINBR 的训练过程图 (图 4-11)、拟合效果图 (图 4-12) 及运行结果。

MSE =
0.0071

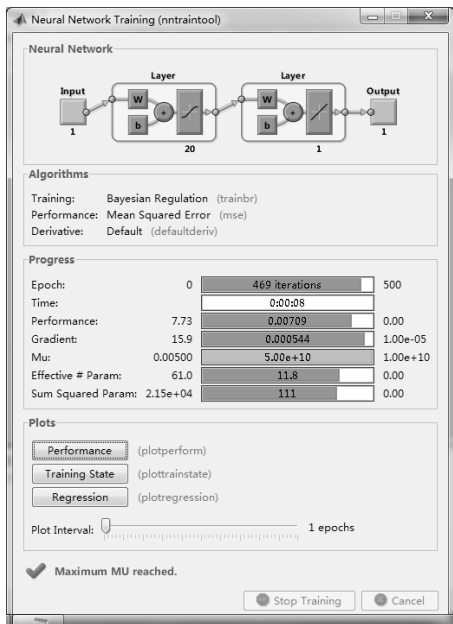


图 4-11 贝叶斯正则化算法训练过程图

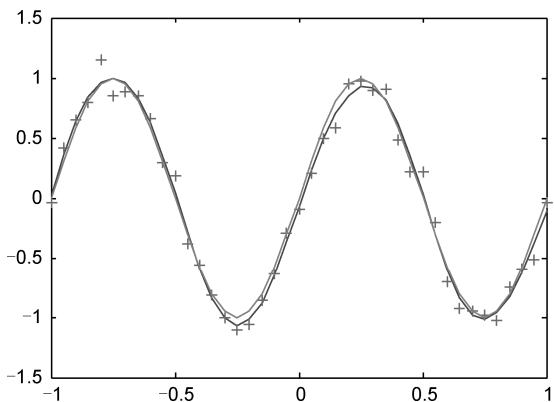


图 4-12 贝叶斯正则化算法拟合图

通过采用两种不同的训练算法，可以得到如图 4-10 和图 4-12 所示的两种拟合结果。图中的实线表示拟合曲线，虚线代表不含白噪声的正弦曲线，“+”点为含有白噪声的正弦样本数据点。显然，经 `trainlm` 函数训练后的神经网络对样本数据点实现了“过度匹配”，而经 `trainbr` 函数训练的神经网络对噪声不敏感，具有较好的推广能力。

第5章 神经网络在选址与地震预测中的应用



本章将举例说明神经网络在实际领域中的应用。

5.1 配送中心选址

合理的物流配送中心选址能节省费用，加快货物的流通，增加物流企业的收益，因此，物流配送中心的选址决策对于整个物流系统的优化是个十分重要的问题。

1. 问题概述

长期以来，对于物流配送中心选址，人们提供了一些决策方法，如模糊综合评判法、AHP层次分析法及结合层次法的模糊排序方法等。但这些方法也有一些缺点，利用模糊综合评判法，其指标权重难以确定；用专家打分法确定权重，人为因素又过重；利用层次分析法确定权重可以弱化人为因素，但是层次分析法要求指标的层次结构系统中的要素互相独立，否则就不能应用这种方法，而这些指标之间往往存在依赖关系，如地价和运输条件、政府政策和经营环境等。

模糊综合评价作为一种多属性的综合评价方法，其隶属函数权重有一定主观性，因此它的应用就有局限性。而BP算法则可以较客观地评价不同的方案，将模糊方法应用到BP算法的输入值中。综合两种算法的优势，通过网络学习得到选址的优化度，进而得到对于选址方案的一个较为准确的评价。

2. 网络实现

物流配送中心的选址通常是在一定的原则（如降低成本原则、经济效益原则、提高客户服务水平原则等）的指导之下，预先选择一些方案，然后再通过各种方法对这些方案进行比较，最终从中选出满意的一个或几个方案作为新中心的地址。影响配送中心选址的因素很多，可以根据物流学的原理，结合自身的实际情况，选择其中较重要的一些因素作为决策指标。这些因素大致可分为自然环境、交通运输条件、经营环境、候选地条件及公共设施几大类。指标选择的好坏对正确决策相当重要。选址主要考虑几个因素：客户的分布、供应商的分布、交通条件、用地条件、自然地理条件和环境条件等。

采用模糊方法计算影响因素的隶属度作为神经网络的输入，在 n 个影响因素中，既有定量因素也有定性因素，对不同的因素要用不同的方法来确定隶属度。对定量因素可以采用常见模糊分布来确定隶属函数，例如，模糊约束集合运输距离最小符合的模糊分布-降半梯形分

布。对于定性因素而言，首先要用专家咨询法、专家评分法等方法进行量化，再利用相关的隶属函数确定隶属度。这里考虑的定量因素有：候选地地价、运输距离和候选地面积。其中运输距离的隶属度通过前面介绍的方法确定，而另两种定量指标中，候选地地价与运输距离的隶属函数相同，模糊约束集合候选地面积适中符合模糊分布-正态分布，对应的隶属函数为：

$$u(x) = e^{-k(x-c)^2} \quad (k, c > 0)$$

而定性因素有：客户分布、供应商分布、地质条件、通信条件、道路设施。定性指标首先对不同的情况进行分类描述，再将自然语言的评价分类转换为相应的隶属度，如表 5-1 所示。

表 5-1 定性指标的隶属函数

指 标	分 类 描 述	模糊约束集	隶 属 度
客户分布	集中, 一般, 分散	客户分布集中 A1	A1={1, 0.5, 0}
供应商分布	集中, 一般, 分散	供应商分布集中 A2	A2={1, 0.5, 0}
通信条件	很好,好,一般, 差, 很差	通信条件良好 A3	A3={1, 0.8, 0.5, 0.2, 0}
地质条件	很好,好,一般, 差, 很差	地质条件良好 A4	A4={1, 0.8, 0.5, 0.2 0}

将指标隶属度输入网络中，将专家值作为网络的期望输出。表 5-2 所示为经过量化和模糊处理，并求出隶属度的实验训练数据。表 5-3 所示为经过初步筛选后的实验数据及其方案的评价结果和排序。

表 5-2 输入样本

方案序号	地质条件	客户分布	候选地 地价	供应商 分布	运输距离	通信条件	候选地 面积	道路设施	专家评价 结果
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.80	0.87	0.89	0.82	0.78	0.80	0.75	0.33	0.79
3	0.67	0.93	0.22	0.75	1.00	0.80	0.49	0.66	0.74
4	0.92	0.80	0.89	0.92	0.89	0.80	1.00	1.00	0.81
5	0.87	0.93	1.00	1.00	1.00	1.00	1.00	1.00	0.96
6	0.80	0.72	0.89	0.82	0.89	0.80	0.75	1.00	0.83
7	0.67	0.72	0.67	0.66	0.67	0.60	0.49	0.66	0.69
8	0.72	0.80	0.78	0.75	0.78	0.80	0.75	0.66	0.75
9	0.60	0.60	0.56	0.58	0.56	0.60	0.49	0.66	0.58
10	0.47	0.47	0.44	0.41	0.44	0.40	0.49	0.35	0.51

表 5-3 测试样本及测试结果

方案 序号	地质条件	客户分布	候选地 地价	供应商 分布	运输距离	通信条件	候选地 面积	道路设施	评价结果	排序
11	0.40	0.40	0.33	0.33	0.33	0.40	0.49	0.35	0.48	4
12	0.08	0.93	0.56	0.92	0.89	0.60	0.24	0.33	0.81	2

续表

方案序号	地质条件	客户分布	候选地地价	供应商分布	运输距离	通信条件	候选地面积	道路设施	评价结果	排序
13	0.67	0.60	0.89	0.82	1.00	0.80	0.75	0.29	0.97	1
14	0.32	0.40	0.67	0.33	0.33	0.80	0.75	0.35	0.54	3
15	0.87	0.72	0.89	0.92	0.89	0.40	0.49	0.29	0.47	5

根据模糊规则，可得到网络的训练样本 P 和 T。完整的 MATLAB 代码如下：

```
clear all;
P=[1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00;
    0.80 0.87 0.89 0.82 0.78 0.80 0.75 0.33;
    0.67 0.93 0.22 0.75 1.00 0.80 0.49 0.66;
    0.92 0.80 0.89 0.92 0.89 0.80 1.00 1.00;
    0.87 0.93 1.00 1.00 1.00 1.00 1.00 1.00;
    0.80 0.72 0.89 0.82 0.89 0.80 0.75 1.00;
    0.67 0.72 0.67 0.66 0.67 0.60 0.49 0.66;
    0.72 0.80 0.78 0.75 0.78 0.80 0.75 0.66;
    0.60 0.60 0.56 0.58 0.56 0.60 0.49 0.66;
    0.47 0.47 0.44 0.41 0.44 0.40 0.49 0.35];
T=[1.00 0.79 0.74 0.81 0.96 0.83 0.69 0.75 0.58 0.51];
%根据 Kolmogorov 定理，由于输入层有 8 个节点，所以中间层有 17 个节点
%中间层神经元的传递函数为 tansig
%输出层有 1 个节点，其神经元传递函数为 logsig
%训练函数采用 trainlm
net=newff(minmax(P),[17,1],{'tansig','logsig'},'trainlm');
%训练步数为 1000 次
%训练目标误差为 0.001
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
net=train(net,P,T);
Y=sim(net,P);
```

得到的训练误差过程如图 5-1 所示。

预测过程代码为：

```
P_test=[0.40 0.40 0.33 0.33 0.33 0.40 0.49 0.35;
    0.08 0.93 0.56 0.92 0.89 0.60 0.24 0.33;
    0.67 0.60 0.89 0.82 1.00 0.80 0.75 0.29;
    0.32 0.40 0.67 0.33 0.33 0.80 0.75 0.35;
    0.87 0.72 0.89 0.92 0.89 0.40 0.49 0.29];
Y=sim(net,P_test)
```

输出结果为：



Y =
0.4820 0.6844 0.8929 0.3673 0.9290

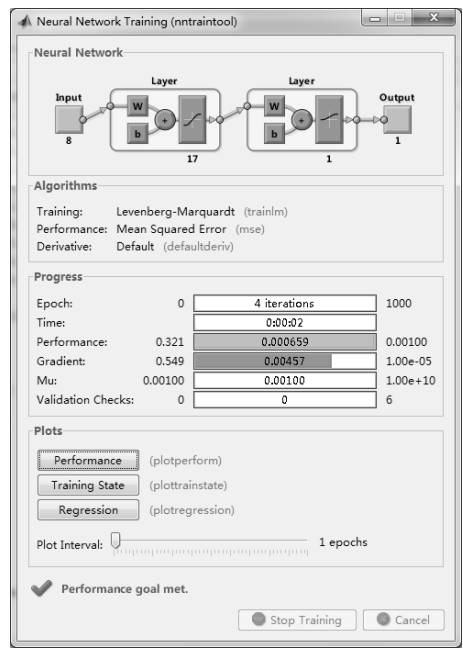


图 5-1 网络训练过程

从以上计算结果可以看出，方案 13 最优，方案 15 最差，这也与各方案的指标因素特点基本一致，根据此结果可进行配送中心选址的决策。

5.2 地震预报

地震预报是地理问题研究领域中的一个重要课题，准确的地震预报可以帮助人们及时采取有效措施，降低人员伤亡和经济损失。引发地震的相关性因素很多，在实际地震预报中，前兆及地震学异常的时间和种类多少与未来地震震级大小有一定关系，但是异常与地震之间有较强的不确定性，同样一种预报方法或前兆在一些地震前可能异常很突出，但在另一些地震前兆则可能表现很不明显甚至根本不出现。同样，一些异常出现后，其后也不一定发生较强地震。因此地震前的各类异常与未来地震的震级及发震时间之间具有较强的非线性关系。这种孕育过程的非线性和认识问题的困难性使得人们很难建立较完善的物理理论模型，并通过某种解析表达式进行表达。而神经网络是一种高度自适应的非线性动力系统，通过 BP 神经网络学习可以得到输入与输出之间的高度非线性映射，因此使用神经网络可以建立起输入与输出之间的非线性关系。

相对于传统的预报方法，神经网络在处理这方面问题中有着独特的优势，主要体现在：

- (1) 容错能力强。由于网络的知识信息采用分布式存储，个别单元的损坏不会引起输出错误。这就使得预测或识别过程中容错能力强，可靠性高。

(2) 预测或识别速度快。训练好的网络在对未知样本进行预测或识别时仅需要少量的加法和乘法，使得其运算速度明显快于其他方法。

(3) 避开了特征因素与判别目标的复杂关系描述，特别是公式的表述。网络可以自己学习和记忆各输入量和输出量之间的关系。

本节将根据某地的地震资料，研究如何利用神经网络工具箱，实现基于神经网络的地震预报。

5.2.1 问题概述

以我国西部某地震常发地区的地震资料作为样本来源，实现基于神经网络的地震预报。根据这些地震资料，提取出 7 个预报因子和实际发生的震级 M 作为输入和目标向量。预报因子为：

- (1) 半年内 $M \geq 3$ 的地震累计频度；
- (2) 半年内能量释放积累值；
- (3) b 值；
- (4) 异常地震群个数；
- (5) 地震条带个数；
- (6) 是否处于活动期内；
- (7) 相关地震区地震震级。

一共收集了 10 个学习样本，如表 5-4 所示。

表 5-4 学习样本

地震累计频度	累计释放能量	b 值	异常地震群个数	地震条带个数	活动周期	相关区震级	实际震级
0	0	0.62	0	0	0	0	0
0.3915	0.4741	0.77	0.5	1		0.3158	0.5313
0.2835	0.5402	0.68	0	1		0.3158	0.5938
0.6210	1.0000	0.63	1	1		1.0000	0.9375
0.4185	0.4183	0.67	0.5	1		0.7368	0.4375
0.2610	0.4948	0.71	0	1		0.2632	0.5000
0.9990	0.0383	0.75	0.5	1		0.9474	1.0000
0.5805	0.4925	0.71	0	0		0.3684	0.3570
0.0810	0.0692	0.76	0	0		0.0526	0.3215
0.3915	0.1230	0.98	0.5	0		0.8974	0.6563

表 5-4 中的前 7 项为学习样本中的输入因子，输出因子为实际震级。利用上表中的学习样本对网络进行训练。在训练前，应该对数据进行归一化处理。表 5-4 中数据已经是归一化后的数据了。



5.2.2 网络设计

还是采用单隐层的 BP 网络进行地震预测。由于输入样本为 7 维的输入向量，因此，输入层一共有 7 个神经元，则中间层应该有 15 个神经元。网络只有 1 个输出数据，则输出层只有 1 个神经元。因此，网络应该为 7×15×1 的结构。按照 BP 网络的一般设计原则，中间层神经元的传递函数为 S 型正切函数。由于输出已被归一化到区间[0,1]中，因此，输出层神经元的传递函数可以设定为 S 型对数函数。利用如下代码创建一个符合上述要求的 BP 网络。

```
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1];
net=newff(threshold,[15,1],{'tansig','logsig'},'traingdx');
```

其中，threshold 设定了网络输入向量的取值范围[0,1]，网络所用的训练函数为 traingdx，该函数以梯度下降法进行学习，并且学习速率是自适应的。

中间层的神经元个数是很难确定的，而这又在很大程度上影响着网络的预测性能。这里首先取 15，然后，观察网络性能；之后，再分别取 10 和 20，并与此时的预测性能进行比较，检验中间层神经元个数对网络性能的影响。当网络的预测误差最小时，网络中间层的神经元数目就是最佳值。

5.2.3 网络训练与测试

对于 5.2.2 节得到的网络，利用表 5-4 中的数据进行训练。训练后的网络才有可能满足实际应用的要求。训练参数的设定如表 5-5 所示，其他参数取默认值。

表 5-5 训练参数

训练次数	训练目标
1000	0.01

网络训练的代码如下。

```
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
%init 函数用于将网络初始化
net=train(net,P,T);
```

变量 P 和 T 分别表示网络的输入向量和目标向量，它们是从表 5-4 中得出的。训练过程如图 5-2 所示。

仿真结果为：

```
Y =
    0.2373    0.5275    0.2848    0.8190    0.6141    0.7020    0.4779
```

可见经过 276 次训练后，网络的目标误差达到要求。

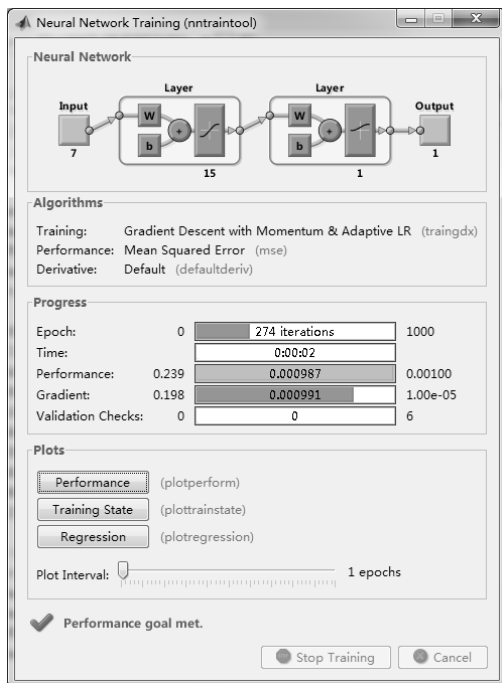


图 5-2 网络训练过程

网络训练结束后，还必须利用另外一组地震数据对其进行测试，数据如表 5-6 所示。所谓测试，实际上是利用仿真函数来获得网络的输出，然后检查输出和实际测量值之间的误差是否满足要求。

测试代码为：

```
P_test=[0.0270 0.0742 0.62 0 0 0 0.2105;
        0.1755 0.3667 0.77 0 0.5 1 0.7368;
        0.4320 0.3970 0.68 0.5 0 1 0.2632;
        0.4995 0.4347 0.63 0 0 1 0.6842;
        0.6885 0.5842 0.67 0.5 0.5 1 0.4211;
        0.5400 0.8038 0.71 0.5 0.5 1 0.5789;
        0.1620 0.2565 0.75 0 0 1 0.4737];
Y=sim(net,P_test)
```

输出结果为：

```
Y =
    0.2373    0.5275    0.2848    0.8190    0.6141    0.7020    0.4779
```

表 5-6 测试数据

地震累计频度	累计释放能量	<i>b</i> 值	异常地震群个数	地震条带个数	活动周期	相关区震级	实际震级
0.0270	0.0742	0.62	0	0	0	0.2105	0.1875
0.1755	0.3667	0.77	0	0.5	1	0.7368	0.4062

续表

地震累计频度	累计释放能量	b 值	异常地震群个数	地震条带个数	活动周期	相关区震级	实际震级
0.4320	0.3790	0.68	0.5	0	1	0.2632	0.4375
0.4995	0.4347	0.63	0	0	1	0.6842	0.5938
0.6885	0.5842	0.67	0.5	0.5	1	0.4211	0.6250
0.5400	0.8038	0.71	0.5	0.5	1	0.5789	0.7187
0.1620	0.2565	0.75	0	0	1	0.4737	0.3750

输出结果经过反归一化处理后得到预报震级，和实际震级相比较可得到网络的预报误差，如表 5-7 所示。

表 5-7 预报误差

实 际 震 级	预 报 震 级	预 报 误 差
4.4	4.5550	0.1550
5.1	5.369	0.0631
5.2	4.9626	0.2374
5.7	5.5387	0.1613
5.8	5.9393	0.1394
6.1	5.9782	0.0218
5.0	4.9258	0.0742

由表 5-7 可见，网络的预报误差比较小，因此，性能可以满足实际应用的要求。预报误差曲线如图 5-3 所示。

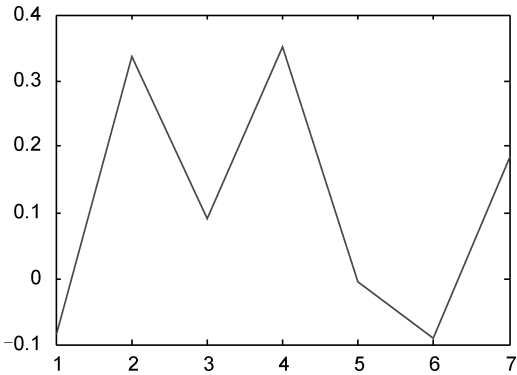


图 5-3 网络的预报误差

接下来将网络中间层数目设置为 10 和 20，并分别进行训练和仿真，得到如下一组训练误差曲线和预报误差曲线。其中，图 5-4 所示为中间层神经元个数为 10 的情况下的训练误差曲线；图 5-5 所示为中间层神经元个数为 20 的情况下的训练误差曲线；

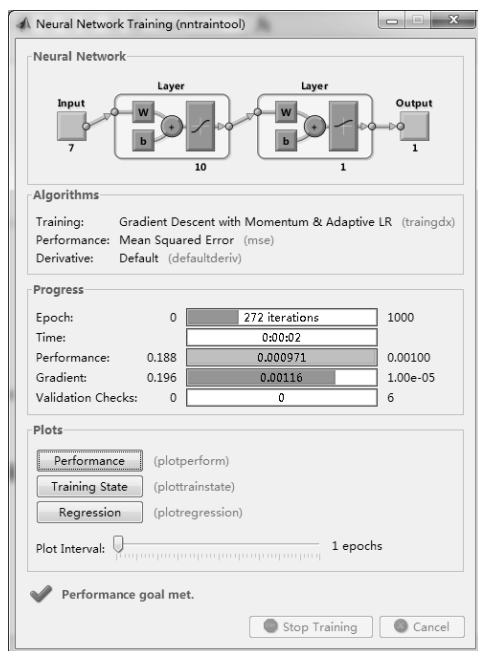


图 5-4 中间层神经元个数为 10 的网络训练过程

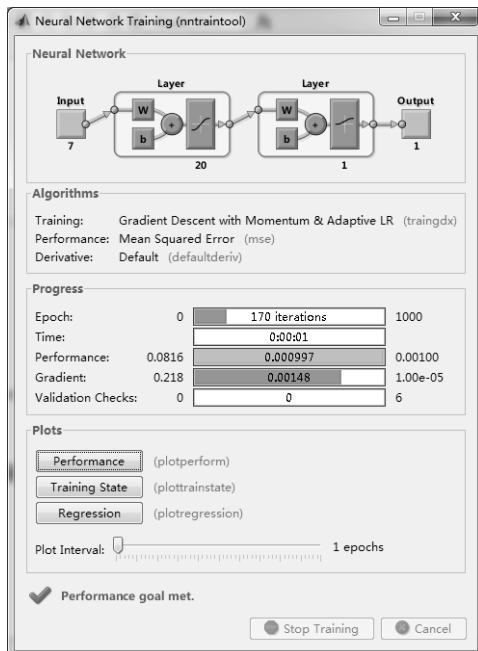


图 5-5 中间层神经元个数为 20 的网络训练过程

此时针对测试数据得到的仿真结果为：

Y =
0.2585 0.5260 0.3561 0.4988 0.6719 0.7060 0.4646

可见，这种情况下网络的预报误差比较大。

中间神经元数目为 20 时，此时网络的仿真结果为：

Y =
0.0933 0.7257 0.7012 0.8313 0.7311 0.5966 0.6434

可见误差非常大。将 3 种情况下的误差进行对比，如图 5-6 所示，可见中间层神经元数为 15 时，网络的预测性能最好。

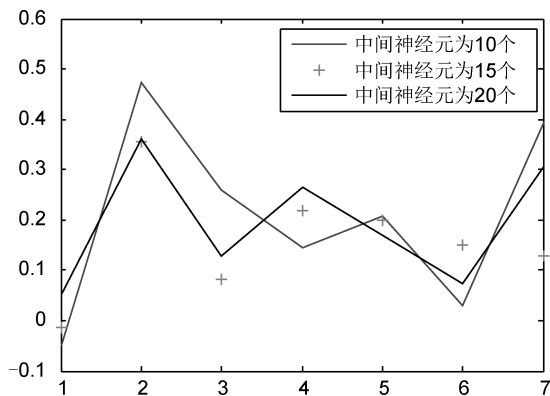


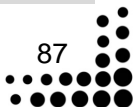
图 5-6 预测误差对比曲线



这也证明了一个重要的结论，就是中间层神经元个数的增加，虽然可以提高网络的映射精度，但并不意味着一定就会提高网络的性能。因此，我们在设计 BP 网络时，不能无限制地增加中间神经元的个数。

完整的 MATLAB 代码为：

```
%输入向量 P 和目标向量 T
P=[0 0 0.62 0 0 0 0;
    0.3915 0.4741 0.77 0.5 0.5 1 0.3158;
    0.2835 0.5402 0.68 0 0.5 1 0.3158;
    0.6210 1.000 0.63 1 .05 1 1.0000;
    0.4185 0.4183 0.67 1 0.5 1 1.0000;
    0.2160 0.4948 0.71 0 0 1 0.2632;
    0.9990 0.0383 0.75 0.5 1 1 0.9474;
    0.5805 0.4925 0.71 0 0 0 0.3684;
    0.0810 0.0692 0.76 0 0 0 0.0526;
    0.3915 0.1230 0.98 0.5 0 0 0.8974]';
T=[0 0.5313 0.5938 0.9375 0.4375 0.5000 1.0000 0.3750 0.3125 0.6563];
%测试向量 P_test 和实际输出 T_test
P_test=[0.0270 0.0742 0.62 0 0 0 0.2105;
    0.1755 0.3667 0.77 0 0.5 1 0.7368;
    0.4320 0.3970 0.68 0.5 0 1 0.2632;
    0.4995 0.4347 0.63 0 0 1 0.6842;
    0.6885 0.5842 0.67 0.5 0.5 1 0.4211;
    0.5400 0.8038 0.71 0.5 0.5 1 0.5789;
    0.1620 0.2565 0.75 0 0 1 0.4737]';
T_test=[0.1875 0.4062 0.4375 0.5938 0.6250 0.7187 0.3570];
threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1];
a=[10 15 20];
for i=1:3
    net=newff(threshold,[a(i),1],{'tansig','logsig'},'traingdx');
    net.trainParam.epochs=1000;
    net.trainParam.goal=0.001;
    %init 函数用于将网络初始化
    net=init(net);
    net=train(net,P,T);
    Y(i,:)=sim(net,P_test);
end
figure;
%绘制误差曲线
%中间层神经元个数为 10
plot(1:7,Y(1,:)-T_test);
hold on;
%中间层神经元个数为 15
```





```
plot(1:7,Y(2,:)-T_test,'r+');  
hold on;  
%中间层神经元个数为 20  
plot(1:7,Y(3,:)-T_test,'k--');  
hold off;  
legend('中间神经元为 10 个','中间神经元为 15 个','中间神经元为 20 个')
```

5.2.4 网络实现

自组织竞争神经网络能够对输入模式进行自组织训练和判断，并将其最终分为不同的类型。与 BP 神经网络方法相比，这种自组织、自适应的学习能力进一步拓宽了人工神经网络在模式识别、分类方面的应用。在地震预报中，有时需要根据各种地震活动性指标（包括前兆指标）将发生在不同时间、空间和强度的地震进行归类研究，根据这些样本的特征对其他样本进行外推预报。分类方法有模糊聚类、投影寻踪和神经网络等。这里采用自组织竞争网络对某地震例进行分类研究。

利用自组织竞争网络进行地震预报，首先应该提取有关地震预报的重要指标，确定网络结构。这里以我国某地及其邻近地区从 1987—1989 年的地震趋势作为检验实例，研究的时间为 1 年，所选的 11 项地震活动指标为：

- 次数最多的地震震级；
- b 值；
- 平均震级；
- 平均纬度；
- 平均纬度偏差；
- 平均经度；
- 平均经度偏差；
- 最大地震震级；
- ML 大于 115 的地震次数；
- 相邻两年的地震次数差。当绝对值超过 8 一个数量级时，先除以 10 再取整；
- 相邻两年最大地震的震级差。当其为负数时进行作乘 0.1 处理。

所有的实际地震数据如表 5-8 所示。利用前 10 年的数据参加竞争训练，最后 1 年的数据作为测试样本。按照震级的大小分为：一般地震、中等地震和严重地震 3 类，因此这里需要设置神经元数目为 3 个。为了加快学习速度，将学习速度设定为 0.1。表 5-8 的数据是已经归一化处理以后的数据。

表 5-8 地震活动指标年值

年份	次数最多的震级	b 值	平均震级	纬度偏差	经度偏差	平均纬度	平均经度	最大震级	次数	次数差	震级差
1978	0.3125	0.45	0.4902	0.7639	0.93	0.4643	0.1765	3.9	0.0473	0.5	0.1
1979	0.3125	0.49	0.3333	0.8611	0.57	0	0	6.3	0.8581	12	2.4



续表

年份	次数最多的震级	b 值	平均震级	纬度偏差	经度偏差	平均纬度	平均经度	最大震级	次数	次数差	震级差
1980	0	0.65	0.7647	1.0000	0.96	0.1876	0.0588	3.5	0.2162	0.03	0.28
1981	0	0.60	0.0196	0.8889	0.94	0.3214	0.1765	3.9	0.1081	0.2	0.4
1982	0.1875	0.50	0.3137	0.5972	0.80	0.1876	0.3529	5.0	0.1419	1.0	1.1
1983	0	0.62	0	0.8194	0.96	1.0000	0.2353	3.5	0	0.2	0.15
1984	1.000	0.36	1.0000	0	0.53	0.1876	1.0000	6.3	1.0000	15	2.8
1985	0.5000	0.43	0.5686	0.1528	0.70	0.1492	0.9412	4.1	1.0000	0	0.22
1986	0.1875	0.42	0.6471	0.7917	1.12	0.2857	0.5882	5.1	0.0405	0.02	1.0
1987	0.5000	0.43	0.6078	0.6528	0.89	0.3214	0.6471	5.4	0.0405	0	0.3
1988	0.3125	0.43	0.6078	0.8333	1.05	0.4286	0.5882	4.0	0.0878	1	0.14
1989	0.3161	0.45	0.5001	0.7853	1	0.4235	0.1825	4.1	0.0501	0.4	0.12

本实例的完整 MATLAB 代码如下。

```
P=[0.3125 0.3125 0 0 0.1875 0 1.0000 0.5000 0.1875 0.5000;
    0.45 0.49 0.65 0.60 0.50 0.62 0.36 0.43 0.42 0.43;
    0.4902 0.3333 0.7647 0.0196 0.3137 0 1.0000 0.5686 0.6471 0.6078;
    0.7639 0.8611 1.0000 0.8889 0.5972 0.8194 0 0.1528 0.7917 0.6528;
    0.93 0.57 0.96 0.94 0.80 0.96 0.53 0.70 1.12 0.89;
    0.4643 0 0.1876 0.3214 0.1876 1.0000 0.1876 0.1492 0.2857 0.3214;
    0.1765 0 0.0588 0.1765 0.3529 0.2353 1.0000 0.9412 0.5882 0.6471;
    3.9 6.3 3.5 3.9 5.0 3.5 6.3 4.1 5.1 5.4;
    0.0473 0.8581 0.2162 0.1081 0.1419 0 1.0000 1.0000 0.0405 0.0405;
    0.5 12 0.03 0.2 1.0 0.2 15 0 0.02 0;
    0.1 2.4 0.28 0.4 1.1 0.15 2.8 0.22 1.0 0.3];
%创建竞争型网络, 竞争层神经元个数为 3, 学习速率为 0.1
net=newc(minmax(P),3,0.1);
%对网络进行训练
net=init(net);
net=train(net,P);
%测试网络
y=sim(net,P);
y=vec2ind(y)
```

训练结果为:

```
TRAINR, Epoch 0/100
TRAINR, Epoch 25/100
TRAINR, Epoch 50/100
TRAINR, Epoch 75/100
TRAINR, Epoch 100/100
```



```
TRAINR, Maximum epoch reached.
```

输出结果为:

```
y =  
    1    2    1    1    3    1    2    3    3    3
```

由此可见,表 5-8 中第 1、3、4、6 行属于第一类,即 1978 年、1980 年、1981 年和 1983 年属于第一类;2、7 行即 1979 年和 1984 年属于第二类。其余的,1982 年、1985 年、1985 年和 1987 年属于第三类。直接检查表中的数据,我们会发现属于同一类的数据是比较相似的,这也验证了以上分类的结果。

利用竞争型网络进行预报的原理为,通过训练样本对网络进行训练,训练好的网络中记忆了所有的分类模式。当输入一个新的样本后,激发了对应的神经元,就可以对新的样本进行分类。在本例中,可以预报地震属于哪一级。

接下来利用网络预测 1989 年地震属于哪一级,实际上这也是对网络进行测试。通过直接进行数据对比,我们认为 1989 年的震级应该和 1978 年的一致,因为两年的数据非常接近。MTALAB 代码如下。

```
P=[0.3125 0.45 0.5001 0.7853 1 0.4235 0.1825 4.1 0.0501 0.4 0.12]';  
y=sim(net,P);  
y=vec2ind(y)
```

输出结果为:

```
y =  
    1
```

由此可见,网络有着比较好的预报精度。

第 6 章 模糊神经网络的算法分析与实现

模糊系统和神经网络控制是智能控制领域内的两个重要分支。模糊系统是仿效人的模糊逻辑思维方法设计的一类系统，这一方法本身就明确地说明了系统在工作过程中允许定性知识的存在。另一方面，神经网络在计算处理信息的过程中所表现出的学习能力和容错性来自于其网络自身的结构特点。

模糊神经网络是一种集模糊逻辑推理的强大结构性知识表达能力与神经网络强大的自学习能力于一体的新技术，它是模糊逻辑推理与神经网络有机结合的产物。

6.1 模糊神经网络的形式

神经网络和模糊控制的结合方式有 3 种。

1. 神经模糊系统

神经模糊系统用神经元网络来实现模糊隶属函数、模糊推理，基本上（本质上）还是 FLN。模糊控制系统是依据一些由人们总结出来的描述各种因素之间相互关系的模糊性语言经验规则，并将这些经验规则上升为简单的数值运算，以便让机器代替人在相应问题面前具体实现这些规则。在一般的模糊系统设计中，规则是由对所解决的问题持有丰富经验的专业人员以语言的方式表达出来的。

模糊神经网络模型如图 6-1 所示。

该模型以模糊控制为主体，应用神经元网络，实现模糊控制的决策过程，以模糊控制方法为“样本”，对神经网络进行离线训练学习。“样本”就是学习的“教师”。所有样本学习完以后，这个神经元网络就是一个聪明、灵活的模糊规则表，具有自学习、自适应功能。

2. 模糊神经网络

神经网络模糊化，本质上还是 ANN，其模型结构如图 6-2 所示。

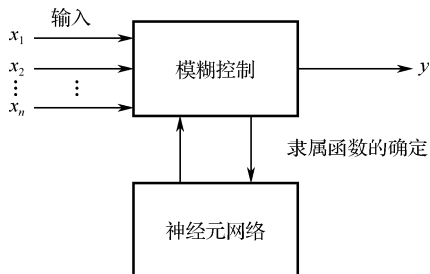


图 6-1 神经模糊系统模型结构图

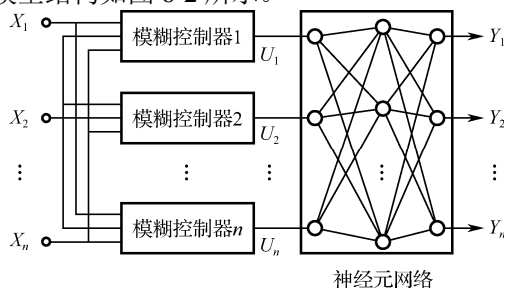


图 6-2 模糊神经网络模型结构图



该模型以神经网络为主体，将输入空间分割成若干不同类型的模糊推论组合，对系统先进行模糊逻辑判断，以模糊控制器输出作为神经元网络的输入。后者具有自学习的智能控制特性。

3. 模糊-神经混合系统

模糊-神经混合系统为二者的有机结合。模型结构如图 6-3 所示。该模型根据输入量的不同性质分别由神经网络与模糊控制直接处理输入信息，并作用于控制对象，更能发挥各自的控制特点。

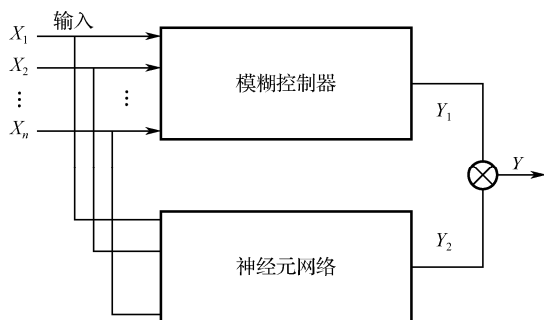


图 6-3 模糊-神经混合系统模型结构图

6.2 神经网络和模糊控制结合的优点

神经网络与模糊控制结合具有如下几个优点：

- (1) 使网络的结构和权值具有了明确的物理意义，网络的结构设计和权值的初始化都有了理论的根据，避免了网络陷入局部最优。
- (2) 可以利用神经网络的学习能力来调整模糊控制的控制规则和模糊化的方式，使模糊控制具有了一定的自适应能力。
- (3) 模糊神经网络将定性的知识表达和定量的数值运算很好地结合了起来，具有很好的控制效果。

6.3 神经模糊控制器

神经模糊控制器是用神经网络构成的模糊控制器，是神经模糊控制的核心。神经模糊控制器最关键的是其结构和学习问题。神经模糊控制器的结构应该是这样一种拓扑结构，它可以处理模糊信息，并能完成模糊推理。神经模糊控制器的学习则是一种能完成模糊推理的神经网络的学习。在这一节中，主要介绍神经模糊控制器的结构及学习算法。

逻辑神经元是模糊神经元的一种，用逻辑神经元可以构成执行 Mamdani 推理的模糊控制器。在这种模糊控制器中，每条控制规则可以用一个模糊神经网络实现。

- (1) 模糊控制系统的参数。考虑一个模糊控制的过程，并且具有图 6-4 中约定的参数。

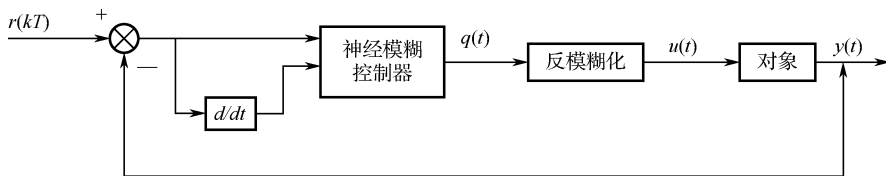


图 6-4 神经模糊控制系统示意图

在时间 t ，系统输入为 $r(t)$ ，输出为 $y(t)$ ，设采样周期为 T ，并在离散时间 $t = T, 2T, \dots$ ，对输出进行检测，则偏差及偏差变化求取如下：

$$e(t) = r(t) - y(t) \quad (6-1)$$

$$\Delta e(t) = \frac{[e(t) - e(t-T)]}{T} \quad (6-2)$$

对于神经模糊控制器，其输入为 $e(t)$ 、 $\Delta e(t)$ ，模糊输出为 $q(t)$ ，反模糊化后的输出为 $u(t)$ 。模糊控制器的控制规则集为：

$$R_r: \text{如果 } e = A_i \text{ 和 } \Delta e = B_j \text{ 那么 } Q = C_p$$

$$1 \leq r \leq g$$

g 表示有 g 条控制规则。

在神经模糊控制器中，关键是如何根据输入 e 、 Δe 得到 Q ，即实现模糊控制规则。

设 $A_i \in [a_1, b_1]$ ， $B_j \in [a_2, b_2]$ ， $C_p \in [a_3, b_3]$ ，则有 e 、 Δe 和 Q 的论域中，选择如下离散值：

- 在 $[a_1, b_1]$ 中取的离散点为 x_i ，并且有：

$$x_0 = a_1 \quad (6-3)$$

$$x_i = a_1 + \frac{i(b_1 - a_1)}{N_1} \quad (6-4)$$

其中， $1 \leq i \leq N_1$ ， N_1 为正整数。

- 在 $[a_2, b_2]$ 中取的离散点为 y_j ，并且有：

$$y_0 = a_2 \quad (6-5)$$

$$y_j = a_2 + \frac{j(b_2 - a_2)}{N_2} \quad (6-6)$$

其中， $1 \leq j \leq N_2$ ， N_2 为整数。

- 在 $[a_3, b_3]$ 中取的离散点为 z_i ，并且有：

$$z_0 = a_3 \quad (6-7)$$

$$z_i = a_3 + \frac{i(b_3 - a_3)}{N_3} \quad (6-8)$$

$1 \leq i \leq N_3$ ， N_3 为正整数。

对于给出的模糊控制规则中的模糊量 A_i ， B_j ， C_p ，其隶属度表示成：

$$A_i(x_j), 0 \leq j \leq N_1$$

$$B_j(y_k), 0 \leq k \leq N_2$$

$$C_p(z_i), 0 \leq i \leq N_3$$

(2) 实现控制规则的神经网络。逻辑神经元组成的模糊神经网络如图 6-5 所示。

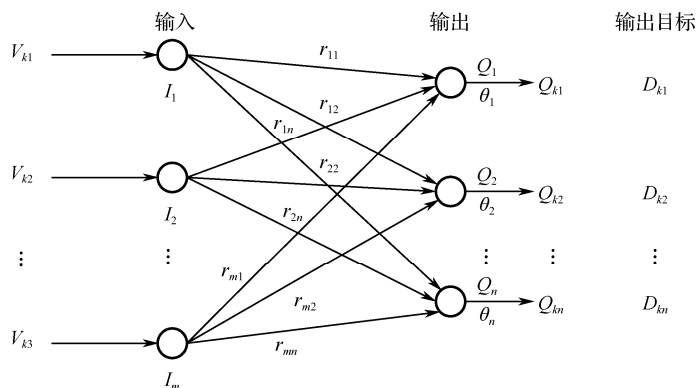


图 6-5 模糊神经网络

它有 m 个输入神经元，分别表示为 I_1, I_2, \dots, I_m ；同时有 n 个输出元，分别表示为 Q_1, Q_2, \dots, Q_n 。设训练数据集为输入 V_k 和目标 D_k ， $1 \leq k \leq K$ ，并且 V_k 和 D_k 都是向量：

$$V_k = (V_{k1}, V_{k2}, \dots, V_{km}) \quad (6-9)$$

$$D_k = (D_{k1}, D_{k2}, \dots, D_{kn}) \quad (6-10)$$

在给定输入为 V_k 时，有输出向量 Q_k ：

$$Q_k = (Q_{k1}, Q_{k2}, \dots, Q_{kn}) \quad (6-11)$$

在输入神经元 I_j 和输出神经元 Q_i 之间的权系数为 r_{ji} ($j=1, 2, \dots, m; i=1, 2, \dots, n$)。输出神经元 Q_i 中有偏置项 θ_i ，用 θ 表示 θ_i 向量，则有：

$$\theta = (\theta_1, \theta_2, \dots, \theta_n) \quad (6-12)$$

在图 6-5 中，神经元是逻辑神经元，故而当给定输入 V_k 后，则有输出：

$$\begin{aligned} Q_{k1} &= [(V_{k1} \wedge r_{11}) \vee (V_{k2} \wedge r_{21}) \vee \dots \vee (V_{km} \wedge r_{m1})] \vee \theta_1 \\ Q_{k2} &= [(V_{k1} \wedge r_{12}) \vee (V_{k2} \wedge r_{22}) \vee \dots \vee (V_{km} \wedge r_{m2})] \vee \theta_2 \\ &\vdots \\ Q_{kn} &= [(V_{k1} \wedge r_{1n}) \vee (V_{k2} \wedge r_{2n}) \vee \dots \vee (V_{km} \wedge r_{mn})] \vee \theta_n \end{aligned}$$

也可以用一个通式表示上面所有的 n 个式子：

$$Q_{ki} = [(V_{k1} \wedge r_{1i}) \vee \dots \vee (V_{km} \wedge r_{mi})] \vee \theta_i, \quad 1 \leq i \leq n \quad (6-13)$$

在图 6-5 中，把输入 V_k 看成一个行向量，把偏置项 θ 看成一个行向量，并令：

$$R = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & & \vdots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{bmatrix} \quad (6-14)$$

则可用向量方程描述神经网络的输入和输出如下：

$$Q_k = (V_k \circ R) \vee \theta \quad (6-15)$$

对于模糊控制器而言，它可以由控制规则集组成。对于控制规则集

R_r ：如果 $e = A_i$ 和 $\Delta e = B_j$ 那么 $Q = C_p$



$$1 \leq r \leq g$$

则可以用一个神经网络模块实现一条规则,用 n 个神经网络模块实现 n 条规则,也就是实现一个模糊控制器。下面说明如何用一个神经网络模块实现一条控制规则。

图 6-5 所示的神经网络可看成一个模块。式 (6-3) 和式 (6-4) 所表示的模糊量 A_i 所取的离散点有 $N_1 + 1$ 个,式 (6-5) 和式 (6-6) 所表示的模糊量 B_j 所取的离散点有 $N_2 + 1$ 个,式 (6-7) 和式 (6-8) 所表示的模糊量 C_p 所取的离散点有 $N_3 + 1$ 个。

$$m = (N_1 + 1) + (N_2 + 1) + N_1 + N_2 + 2 \quad (6-16)$$

$$n = N_3 + 1 \quad (6-17)$$

在输入的 m 个神经元中,把其中的 $N_1 + 1$ 个神经元用于输入 A_i ,其中的 $N_2 + 1$ 个神经元用于输入 B_j ;而输出的 n 个神经元用于目标 C_p ,并以此进行学习,则可以在学习结束时得到能实现一条规则的一个神经网络模块。学习时

$$\begin{aligned} & A_i(X_0) - I_1 \\ & A_i(X_1) - I_2 \\ & \vdots \\ & A_i(X_{N_1}) - I_{N_1 + 1} \end{aligned}$$

而同时

$$\begin{aligned} & B_j(Y_0) - I_{N_1 + 2} \\ & B_j(Y_1) - I_{N_1 + 3} \\ & \vdots \\ & B_j(Y_{N_2}) - I_m \end{aligned}$$

并且,输出目标为:

$$\begin{aligned} & C_p(Z_0) - Q_1 \\ & C_p(Z_1) - Q_2 \\ & \vdots \\ & C_p(Z_{N_3}) - Q_n \end{aligned}$$

在控制规则集中,一共有 g 条规则,最后用 g 个神经网络模块就可以实现模糊控制器,这个控制器即神经模糊控制器。

6.4 神经模糊控制器的学习算法

图 6-5 所示的逻辑神经网络所组成的神经模糊控制器可以用改进的梯度法进行学习。设给出的训练输入为 V_k , 目标为 D_k , $1 \leq k \leq K$, 并且

$$V_k = (V_{k1}, V_{k2}, \dots, V_{km})$$

$$D_k = (D_{k1}, D_{k2}, \dots, D_{kn})$$

而图 6-5 所示的网络要输入为 V_k 时,实际输出为 Q_k 。学习的目的是令 Q_k 逼近于 D_k 。实质上,



就是修改网络的权系数 r_{ji} 使得网络的输出 Q_k 趋于目标 D_k 。换言之，是使式 (6-15) 中的 $Q_k = D_k$ 时，求其解 R 。即在

$$D_k = (V_k \circ R) \vee \theta \quad (6-18)$$

中，给出 D_k 、 V_k 及 θ ，求 R 。

为了进行学习，取目标函数 J ：

$$J = \frac{1}{2} \sum_{k=1}^K \sum_{i=1}^n (Q_{ki} - D_{ki})^2 \quad (6-19)$$

学习算法采用改进的梯度算法，这个算法用下面两个公式表示：

$$r_{ji}(l+1) = r_{ji}(l) + \eta_1 \Delta r_{ji}(l+1) + a_1 \Delta r_{ji}(l) \quad (6-20)$$

$$\theta_i(l+1) = \theta_i(l) + \eta_2 \Delta \theta_i(l+1) + a_2 \Delta \theta_i(l) \quad (6-21)$$

其中， $l=1,2,\dots$ 为迭代次数； $\eta_i (i=1,2)$ 是学习速率； $a_i (i=1,2)$ 是动量常数。

在式 (6-20) 和式 (6-21) 中， Δr_{ji} 和 $\Delta \theta_i$ 的意义如下：

$$\Delta r_{ji} = \frac{\partial J}{\partial r_{ji}} \quad (6-22)$$

$$\Delta \theta_i = \frac{\partial J}{\partial \theta_i} \quad (6-23)$$

对于 Δr_{ji} ，可计算如下：

$$\Delta r_{ji} = \sum_{k=1}^K (Q_{ki} - D_{ki}) \frac{\partial Q_{ki}}{\partial r_{ji}} \quad (6-24)$$

$$\frac{\partial Q_{ki}}{\partial r_{ji}} = \begin{cases} 1, & Q_{ki} = r_{ji} \\ 0, & Q_{ki} \neq r_{ji} \end{cases} \quad (6-25)$$

对于式 (6-25)，可以解释如下：假定 $m=2$ ， $n=1$ ，同时忽略下标 k ，则输入有 V_1 、 V_2 ，对于 V_1 有权系数 r_1 ，对于 V_2 有权系数 r_2 ，输出偏置项为 θ 。

设 $V_1=0.7$ ， $V_2=0.3$ ， $r_1=1$ ， $r_2=0.5$ ， $\theta=0$ ，据式 (6-25) 有：

$$\begin{aligned} Q &= (V \circ R) \vee \theta \\ &= \left[(0.7 \quad 0.3) \circ \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} \right] \vee 0 \\ &= [(0.7 \wedge 1) \vee (0.3 \wedge 0.5)] \vee 0 \\ &= 0.7 \end{aligned}$$

故而有

$$\begin{aligned} \frac{\partial Q}{\partial r_1} &= \frac{\partial 0.7}{\partial 1} = 0 \\ \frac{\partial Q}{\partial r_2} &= \frac{\partial 0.7}{\partial 0.5} = 0 \end{aligned}$$

若设 $V_1=0.7$ ， $V_2=0.3$ ， $r_1=0.6$ ， $r_2=0.5$ ， $\theta=0$ ，则根据式 (6-25) 有：



$$\begin{aligned}
 Q &= (V \circ R) \vee \theta \\
 &= \left[(0.7 \quad 0.3) \circ \begin{pmatrix} 0.6 \\ 0.5 \end{pmatrix} \right] \vee 0 \\
 &= [(0.7 \wedge 0.6) \vee (0.3 \wedge 0.5)] \vee 0 \\
 &= 0.6
 \end{aligned}$$

故而有

$$\begin{aligned}
 \frac{\partial Q}{\partial r_1} &= \frac{\partial 0.6}{\partial 0.6} = 0 \\
 \frac{\partial Q}{\partial r_2} &= \frac{\partial 0.6}{\partial 0.5} = 0
 \end{aligned}$$

很明显，式(6-25)的意义是恰当的。

同样，对于 $\Delta\theta_i$ ，可计算如下：

$$\Delta\theta_i = \sum_{k=1}^K (Q_{ki} - D_{ki}) \frac{\partial Q_{ki}}{\partial \theta_i} \quad (6-26)$$

$$\frac{\partial Q_{ki}}{\partial \theta_i} = \begin{cases} 1, & Q_{ki} = \theta_i \\ 0, & Q_{ki} \neq \theta_i \end{cases} \quad (6-27)$$

为了保证 $r_{ji}(l+1)$ 和 $\theta_i(k+1)$ 的值能处于 $[0, 1]$ 区间，故而给出附加的约束条件如下：

(1) 在式(6-20)中，如果式子右边得到的数值小于 0，则令 $r_{ji}(l+1)=0$ ；如果式子右边得到的数值大于 1，则 $r_{ji}(l+1)=1$ 。

(2) 在式(6-21)中，如果式子右边得到的数值小于 0，则令 $\theta_i(k+1)=0$ ；如果式子右边得到的数值大于 1，则 $\theta_i(k+1)=1$ 。

式(6-20)和式(6-21)所表示的学习算法对所选择的 r_{ji} 的初值较为敏感。在初值选择时一般选 $\theta_i=0$ 。

6.5 模糊神经网络 MATLAB 函数

Takagi-Sugeno 型模糊推理具有计算简单，利于数学分析的优点，且易于和 PID 控制方法以及优化、自适应方法结合，从而实现具有优化与自适应能力的控制器或模糊建模工具。

在 MATLAB 模糊逻辑工具箱中，提供了有关对模糊神经系统建模和初始化模糊推理系统的函数。

6.5.1 模糊神经系统的建模函数

在 MATLAB 模糊逻辑工具箱中，提供了对基于 Takagi-Sugeno 型模糊推理的模糊神经系统建模的方法，该模糊推理系统利用反向传播算法和最小二乘算法来完成对输入/输出数据对的建模。相应的函数为 `anfis`，该函数的输出为一个三维或五维向量。当未指定检验数据时，



输出向量为三维。anfis 支持采用输出加权平均的一阶 Takagi-Sugeno 型模糊推理，anfis 函数的调用格式为：

```
[fis,error,stepsize] = anfis(trnData)
[fis,error,stepsize] = anfis(trnData,initFis)
[fis,error,stepsize] = anfis(trnData,numMFs)
[fis,error,stepsize,chkFis,chkErr]=anfis(trnData,initFis,trnOpt,dispOpt,chkData,optMethod)
[fis,error,stepsize,chkFis,chkErr]=...
anfis(trnData,numMFs,trnOpt,dispOpt,chkData,optMethod)
```

其中，fis 参数为学习完成后得到的对应最小均方根误差的模糊推理系统矩阵；error 为训练的均方根误差向量；stepsize 为训练步长向量。当指定检验数据后，输出向量为五维参数向量，chkFis 参数为对检验数据具有最小均方根误差的模糊推理系统，chkErr 为检验数据对应的最小均方根误差向量。

在函数的输入参数向量中，trnData 为用于训练学习的输入输出数据矩阵。该矩阵的每一行对应一组输入输出数据，其中最后一列为输出数据。由于 MATLAB 的自适应神经模糊模型只运行单输出变量的模糊系统，所以 trnData 的维数为训练数据数，列数为输入变量数加 1（输入变量加输出变量数）。

initFis 是用于指定初始的模糊推理系统参数（包括隶属度函数类型和参数）的矩阵，该矩阵可使用函数 fuzzy 通过模糊推理系统编辑器生成，也可使用函数 genfis1 由训练数据直接生成。函数 genfis1 的功能是采用网格分割法生成模糊推理系统，其使用方法参见下面的说明。如果没有指明该参数，函数 anfis 会自动先调用 genfis1 来生成一个默认初始 FIS 推理系统参数。如果调用 anfis 时只使用一个参数即 trnData，genfis1 则使用默认 FIS 结构的每个变量的两条高斯型的隶属度函数，如果 initFis 参数指定的是一个数值或是一个与输入变量个数相同的向量，则系统把这个数值或向量中的对应数值作为相应的输入变量分别的隶属度函数传入函数 genfis1，以生成相应的初始 FIS 系统。关于函数 genfis1，后面将进行介绍。

函数 anfis 的输入参数中，trnOpt 和 dispOpt 分别用于指定训练的有关选项和在训练执行过程中 MATLAB 命令窗口的显示选项。参数 trnOpt 为一个五维向量，其各个分量的定义如下所述。

- trnOpt (1): 训练的次数，默认为 10；
- trnOpt (2): 期望误差，默认为 0；
- trnOpt (3): 初始步长，默认为 0.01；
- trnOpt (4): 步长递减速率，默认为 0.9；
- trnOpt (5): 步长递增速率，默认为 1.1。

如果 trnOpt 的任一个分量为 NaN（非数值：IEEE 的标准缩写）或被省略，则训练采用默认参数。学习训练的过程在训练参数得到指定值或训练误差得到期望误差时停止。训练过程中的步长调整采用如下的策略：

- 当误差连续四次减小时，则增加步长。
- 当误差连续两次出现振荡，即一次增加和一次减少交替发生时，则减小步长。

TrnOpt 的第四和第五个参数分别按照上述策略控制训练步长的调整。

参数 dispOpt 用于控制训练过程中 MATLAB 命令窗口的显示内容，其有四个参数，分别



定义如下。

- dispOpt (1): 显示 ANFIS 的信息, 默认为 1;
- dispOpt (2): 显示误差测量, 默认为 1;
- dispOpt (3): 显示训练步长, 默认为 1;
- dispOpt (4): 显示最终结果, 默认为 1。

当上述某一分量为 0 时, 则不显示相应的内容; 如果为 1、NaN 或省略, 则显示相应内容。

函数 `anfis` 的另一个输入参数为 `chkData`, 该参数为一个与训练数据矩阵有相同列数的矩阵, 用于提供检验数据。当提供检验数据时, ANFIS 返回对于核对数据具有最小均方根误差的模糊推理系统 `fismat2`。

函数最后一个可选的参数是 `optMethod`, 指明网络的训练算法, 可以取 1 和 0。当取 1 时选用 (hybrid) 混合算法, 取 0 时采用反射传播算法 (backpropagation), 默认时采用 hybrid 算法, 也就是最小二乘的方向传播算法。

当函数的训练次数达到或是误差精度目标达到, 就停止训练。

当输入的某个参数为 NaN 或是空矩阵时, 该参数取默认值。注意, 如果想默认前面的参数而使用后面的某个参数时, 则前面的被默认的参数应当用 NaNs 来替代。例如 `[fis,error,stepsize,chkFis,chkErr]==anfis(trnData,initFis,NaN,NaN,chkData)`。

【例 6-1】 利用 MATLAB 实现模糊推理系统的设计。

```
>> clear all;
x = (0:0.1:10)';
y = sin(2*x)./exp(x/5);
trnData = [x y];
numMFs = 5;
mfType = 'gbellmf';
epoch_n = 20;
in_fis = genfis1(trnData,numMFs,mfType);
out_fis = anfis(trnData,in_fis,20);
plot(x,y,'r',x,evalfis(x,out_fis));
legend('训练数据','ANFIS 输出');
```

运行程序, 输出如下, 效果如图 6-6 所示。

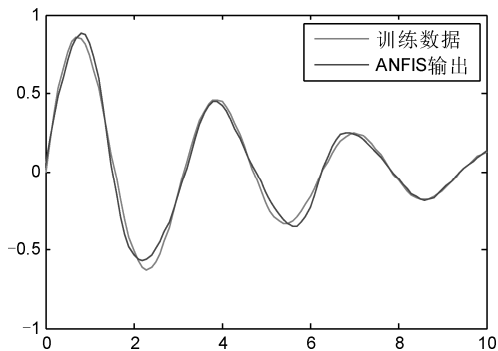


图 6-6 训练数据和 `anfis` 输出曲线效果图



ANFIS info:

Number of nodes: 24
Number of linear parameters: 10
Number of nonlinear parameters: 15
Total number of parameters: 25
Number of training data pairs: 101
Number of checking data pairs: 0
Number of fuzzy rules: 5

Start training ANFIS ...

1	0.0694086
2	0.0680259
3	0.066663
4	0.0653198
5	0.0639961

Step size increases to 0.011000 after epoch 5.

6	0.0626917
7	0.0612787
8	0.0598881
9	0.0585193

Step size increases to 0.012100 after epoch 9.

10	0.0571712
11	0.0557113
12	0.0542741
13	0.052858

Step size increases to 0.013310 after epoch 13.

14	0.0514612
15	0.0499449
16	0.0484475
17	0.0469667

Step size increases to 0.014641 after epoch 17.

18	0.0455003
19	0.0439022
20	0.0423184

Designated epoch number reached --> ANFIS training completed at epoch 20.

【例 6-2】 使用 anfis 函数对给定的数据进行神经模糊建模。

```
>> clear all;  
numMFs=5; % 隶属度函数个数  
mfType='gbellmf'; % 隶属度函数类型  
numEpochs=40; % 训练次数为 40  
numPts=51;  
x1=linspace(0,1,numPts);
```



```

y=0.6*sin(pi*x1)+0.3*sin(3*pi*x1)+0.1*sin(5*pi*x1);
data=[x1' y'];           % 整个数据集
trnData=data(1:2:numPts,:); % 训练数据集
chkData=data(2:2:numPts,:); % 检验数据集
fisMat=genfis1(trnData,numMFs,mfType);
[fis1,truErr,ss,fis2,chkErr]=anfis(trnData,fisMat,numEpochs,NaN,chkData);
% 计算训练后神经模糊系统的输出与训练数据的均方根误差
trnOut=evalfis(trnData(:,1),fis1);
trnRMSE=norm(trnOut-trnData(:,2))/sqrt(length(trnOut));
% 绘制训练过程中均方根误差的变化情况，如图 6-7 所示

```

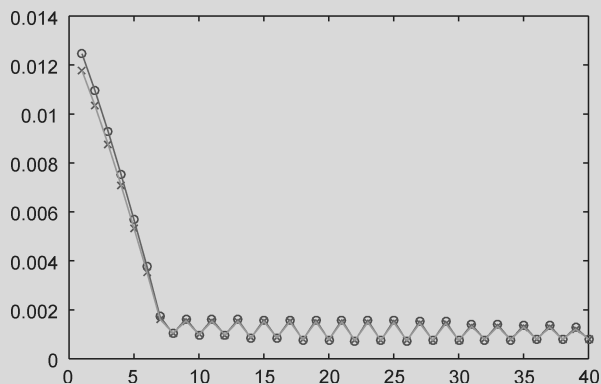


图 6-7 训练过程中均方根误差的变化

```

epoch=1:numEpochs;
plot(epoch,truErr,'o',epoch,chkErr,'mx');
hold on;
plot(epoch,[truErr,chkErr]);
hold off;
>>% 绘制训练过程中步长的变化情况，如图 6-8 所示
plot(epoch,ss,'-',epoch,ss,'x');
xlabel('epochs');ylabel('ss');title('步长');

```

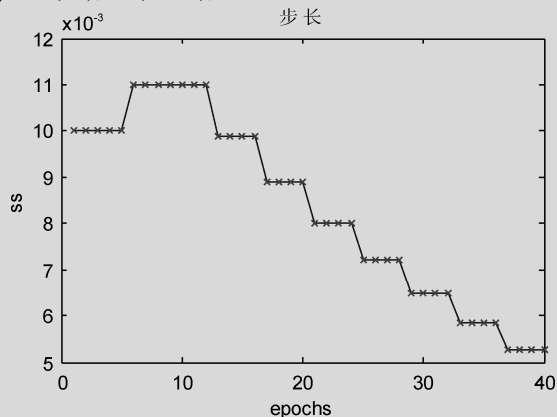
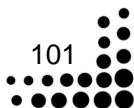


图 6-8 训练步长变化曲线图





```
>>%绘制训练后模糊推理系统的输出曲线，如图 6-9 所示
```

```
[x,mf]=plotmf(fis1,'input',1);plot(x,mf)
```

```
title('fial 成员函数');
```

```
anfis_y=evalfis(x1,fis1);
```

```
plot(x1,y,'-',x1,anfis_y,'mx');
```

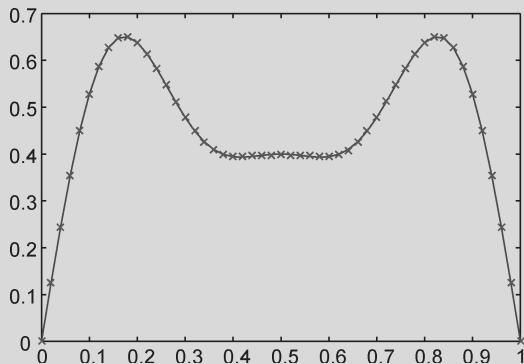


图 6-9 模糊推理系统的输出曲线

6.5.2 采用网格分割方式生成模糊推理系统函数

在给定输入/输出数据后，要利用 `anfis` 函数进行神经模糊系统建模，必须提供一个以输入/输出数据为基础的初始模糊推理系统。函数 `genfis1` 采用网格分割的方式，根据给定数据集生成一个模糊推理系统，因而可以与函数 `anfis` 配合使用。由 `genfis1` 生成的模糊推理系统的输入和隶属度函数的类型及数目可以在使用时指定，也可以采用默认值。函数 `genfis1` 的调用格式为：

```
fismat = genfis1(data)
```

```
fismat = genfis1(data,numMFs,inmftype,outmftype)
```

该函数是用于建立一个初始 **Sugeno** 型模糊系统供函数 `anfis` 训练使用，使用的是网格分割法而不同于 `genfis2` 的模糊聚类法。

在输入参数中，`data` 为给定的输入/输出的训练数据集合。

`numMFs` 为一个整数向量，用于指定输入变量的隶属度函数个数，可以用一个数值表示所有输入变量具有相同数目的隶属度函数。如果是向量，则分别指明每一个输入变量的隶属度函数个数。

参数 `mfType` 用于指定隶属度函数的类型，为字符串数组（分别指明输入变量的隶属度函数类型）或是单个字符串（所有变量使用同种隶属度函数类型）。

`Outmftype` 用于指定输出（MATLAB 的自适应神经模糊模型只支持一个输出变量）的隶属度函数类型，取值可以是“constant”或“linear”。

`fismat` 为生成的模糊推理系统矩阵。当仅使用一个输入参数而不指定隶属度函数的个数和类型时，将使用默认值，即隶属度函数个数为 2，类型为钟型曲线。该函数生成的系统总规则数等于所有输入变量的隶属度函数个数的乘积。例如，三个输入变量每个都有两个隶属度



函数，则规则总数为 $2 \times 2 \times 2 = 8$ 条。

【例 6-3】 利用函数 `genfis1` 产生一个二输入单输出（输入变量分别为 8 条隶属度函数和 4 条隶属度函数，产生 32 条模糊规则）。

实现的 MATLAB 程序代码如下：

```
data=[rand(10,1) 10*rand(10,1)-5 rand(10,1)];
fis=genfis1(data,[8,4],char('pimf','trimf'));
[x, mf]=plotmf(fis,'input',1);
subplot(2,1,1);plot(x,mf);
xlabel('input1 (pimf)');
[x, mf]=plotmf(fis,'input',2);
subplot(2,1,2);plot(x,mf);
xlabel('input2 (trimf)')
```

执行程序代码，输出隶属度函数曲线如图 6-10 所示。

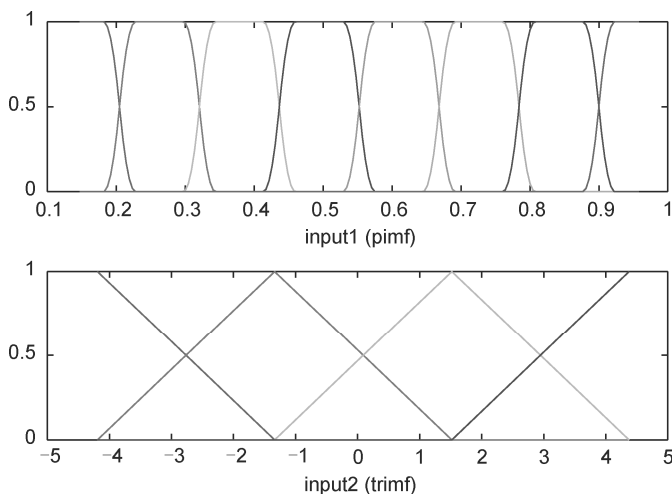


图 6-10 `genfis1` 生成的输入隶属度函数曲线

6.6 MATLAB 模糊神经推理系统的图形用户界面

为了进一步方便用户，在模糊逻辑工具箱中提供了建立模糊神经推理系统的图形界面工具，该工具以交互式图形界面的形式集成了建立、训练和测试神经模糊推理系统等各种功能。要启动该工具，只需在 MATLAB 命令窗口中输入 `anfisedit`，可弹出如图 6-11 所示的图形窗口界面。

该图形界面包括的功能主要有加载数据（Load Data）、生成模糊推理系统（Generate FIS）、训练神经模糊推理系统（Train FIS）和测试神经模糊推理系统（Test FIS）。

1) 加载数据

通过界面上的检查框可以选择加载数据的类型，如训练数据、测试数据、核对数据和演

示数据等。为方便说明，在此选择加载演示数据。加载了 MATLAB 自带的 mgdata.dat 数据，其图形窗口如图 6-12 所示。在图 6-12 的上部显示了数据的变化情况。

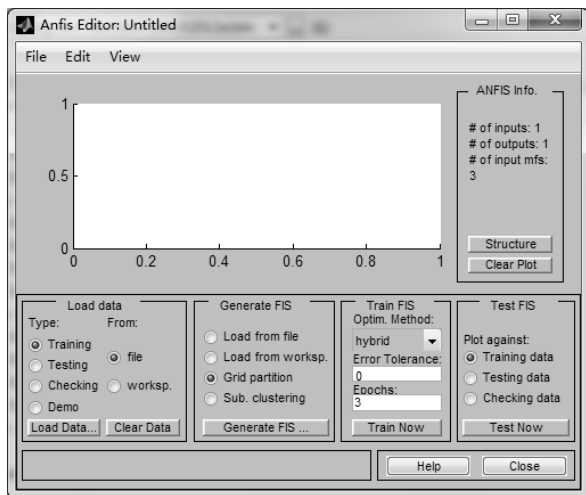


图 6-11 anfis 图形窗口界面

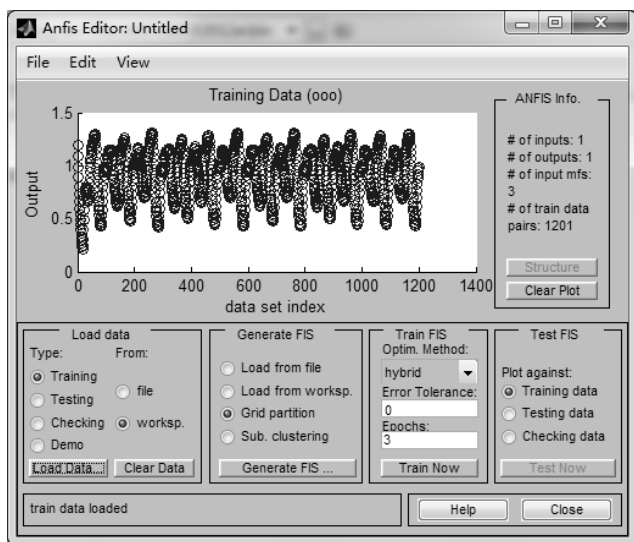


图 6-12 加载演示数据的 anfis 的图形窗口界面

2) 生成模糊推理系统

在生成模糊推理系统时，也可通过检查框选择模糊推理系统的生成方法，如网格分割法（Grid partition）、减法聚类法（Sub.clustering）等。当用鼠标单击生成模糊推理系统的功能按钮时，系统即弹出一个对话框，要求指定模糊推理系统的有关信息，如图 6-13 所示。

在图 6-13 所示的对话框中，要求输入的信息包括输入语言变量隶属度函数的数目、类型和输出隶属度函数的类型等。单击“OK”按钮后，自动生成一个以 anfis 命名的结构。

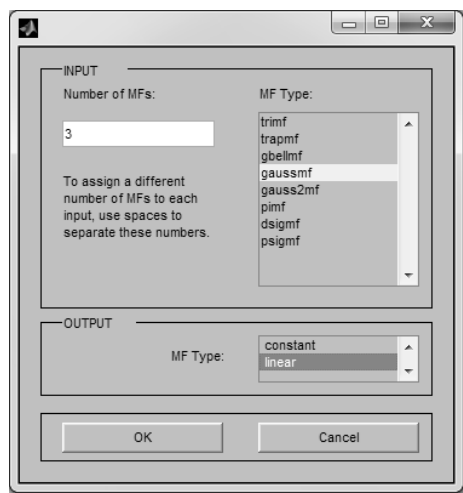


图 6-13 模糊推理系统的对话框

3) 训练神经模糊推理系统

在进行神经模糊推理系统的训练前，可指定优化的方法及有关的优化控制参数。对于前面加载的演示数据，在进行 **anfis** 的训练后，窗口的上部显示了优化过程中误差的变化情况，如图 6-14 所示。

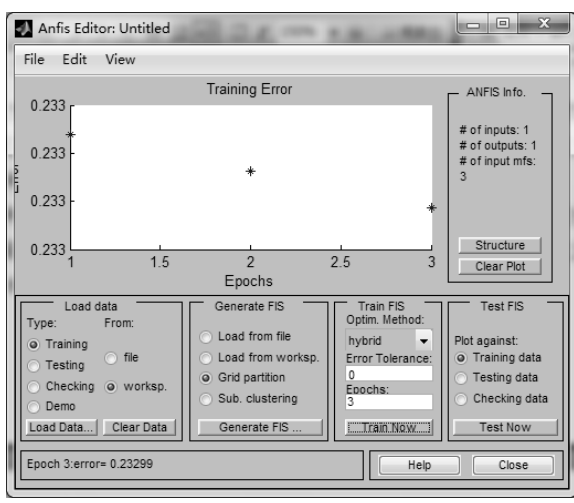


图 6-14 **anfis** 的训练图形界面图

4) 测试 **anfis**

在 **anfis** 的图形界面中，利用右上角的“**Structure**”按钮，可以方便地查看训练得到的 **anfis** 网络结构。对应上述演示数据的神经网络结构如图 6-15 所示。

在完成对 **anfis** 的训练后，可进一步对其进行测试，测试数据可以被指定为训练数据或另外提供的训练数据。测试完成后，将在图形界面的上部显示测试的结果，即 **anfis** 输出数据与

训练数据的比较，在左下方的信息显示这些数据的平均误差，如图 6-16 所示。

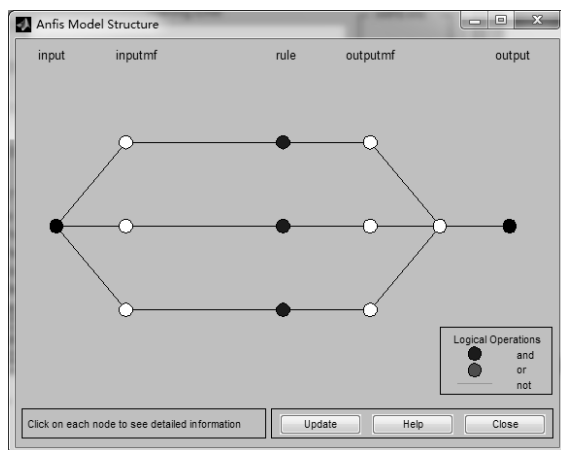


图 6-15 anfis 的神经网络结构

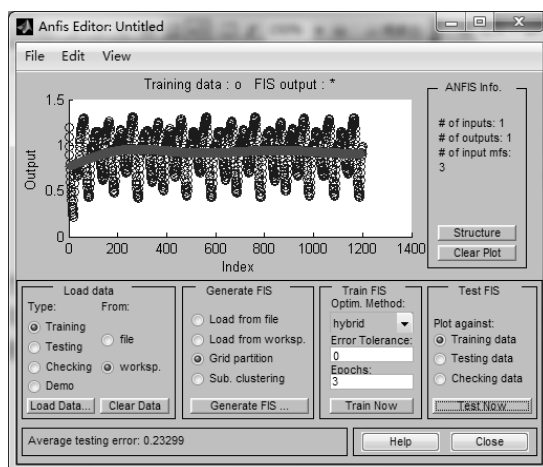


图 6-16 anfis 的输出数据与训练数据的比较

第7章 BP网络的典型应用

在神经网络的训练中，可能出现所谓“过适配”问题，对于训练集的样本其误差可以很小，但对于训练集以外的新样本数据其误差会很大。网络记忆了训练过的样本，但缺乏对新样本泛化的能力。

一种提高网络泛化能力的方法是调整网络的规模，使之刚好足以“适配”。所设计的网络规模越大，网络的函数映射功能越强，那么只要采用的网络规模足够小，即可消除“过适配”现象。在 MATLAB 命令窗口中运行 `nnd11gn` 示例，可看到如何减小网络规模，以避免“过适配”的情况，效果如图 7-1 所示。

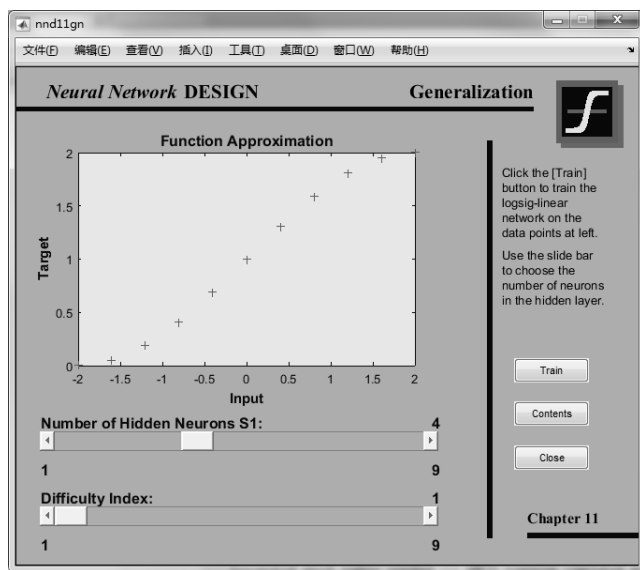


图 7-1 nnd11gn 窗口

遗憾的是，对于一个特定的问题，很难预先知道究竟其网络规模设计为多大是合适的。在神经网络工具箱里，提出了另外两种提高网络泛化能力的方法，即归一化和提前终止法。

7.1 数据归一化方法

数据归一化方法是神经网络预测前对数据常做的一种处理方法。数据归一化处理把所有数据都转化为[0,1]之间的数，其目的是取消各维数据间数量级差别，避免因输入、输出数据数量级差别较大而造成网络预测误差较大。数据归一化的方法主要有以下两种。



(1) 最大最小法，函数如下：

$$x_k = \frac{x_k - x_{\min}}{x_{\max} - x_{\min}}$$

其中， x_{\min} 为数据序列中的最小数； x_{\max} 为序列中的最大数。

(2) 平均数方差法，函数如下：

$$x_k = \frac{x_k - x_{\text{mean}}}{x_{\text{var}}}$$

其中， x_{mean} 为数据序列的均值； x_{var} 为数据的方差。

归一化函数采用 MATLAB 自带的函数 `mapminmax`。该函数有多种形式，常用方法为：

```
[Y,PS] = mapminmax(X,YMIN,YMAX)
```

```
[Y,PS] = mapminmax(X,FP)
```

其中， X 为训练输入、输出原始数据，可为 $N \times Q$ 矩阵或为 $1 \times TS$ 的元胞数值 $N \times Q$ 的矩阵阵列。

$YMIN$ 为阵列每一行的最小取值，默认值为 -1。

$YMAX$ 为阵列的每一行的最大取值，默认值为 1。

FP 作为一个结构参数： $FP.ymin$ ， $FP.ymax$ 。

Y 为归一化后的数据。

PS 为数据归一化后得到的结构体，里面包含了数据最大值、最小值和平均值等信息，可用于测试数据归一化和反归一化。

测试数据归一化和反归一化程序为：

$Y = \text{mapminmax}('apply', X, PS)$ ：测试输入数据归一化。

$r = \text{mapminmax}('reverse', R, ps)$ ：网络预测数据反归一化。

其中，

Y 为预测输入数据。

X 为归一化后的预测数据。

'apply' 表示根据 PS 的值对 X 进行归一化。

R 为网络预测结果。

Ps 为训练输出数据归一化得到的结构体。

r 为反归一化之后的网络预测输出。

'reverse' 表示对数据进行反归一化。

【例 7-1】 对给定数据实现归一化处理。

```
>> clear all;
%使用格式化，使矩阵 x1 每一行的最低和最高值映射到默认区间[-1, +1]。
x1 = [1 2 4; 1 1 1; 3 2 2; 0 0 0];
[y1,PS] = mapminmax(x1)
%应用相同的处理设置新值
x2 = [5 2 3; 1 1 1; 6 7 3; 0 0 0];
y2 = mapminmax('apply',x2,PS)
%利用反归一化再次得到 X1, Y1 处理。
x1_again = mapminmax('reverse',y1,PS)
```



运行程序，输出如下：

```

y1 =
    -1.0000    -0.3333     1.0000
     1.0000     1.0000     1.0000
     1.0000    -1.0000    -1.0000
         0         0         0

PS =
    name: 'mapminmax'
    xrows: 4
    xmax: [4x1 double]
    xmin: [4x1 double]
    xrange: [4x1 double]
    yrows: 4
    ymax: 1
    ymin: -1
    yrange: 2
    no_change: 0

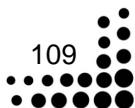
y2 =
     1.6667    -0.3333     0.3333
     1.0000     1.0000     1.0000
     7.0000     9.0000     1.0000
         0         0         0

x1_again =
     1     2     4
     1     1     1
     3     2     2
     0     0     0
  
```

7.2 提前终止法

在提前终止算法中，能够利用的数据被分为 3 个子集，第一个子集是训练样本集，用于计算梯度和修正网络的权值及阈值。第二个子集是确认样本集，在训练的过程中，监控确认样本集的误差。在训练的最初阶段，确认样本集的误差在正常地减小，即训练样本集的误差；而在网络开始出现“过适配”时，确认样本集的误差在明显增加，当确认样本集的误差连续增加的次数达到指定的迭代次数时，训练便被终止，此时，网络返回具有最小确认样本集误差的权值和阈值。第三个子集是测试样本集，但在训练过程中并没有用到测试样本集的误差，而是用于各种不同模型的比较；另外，在训练期间画出测试样本集的误差曲线也是有用的，如果测试样本集的误差与确认样本集误差间的误差也达到最小值时，所需的迭代次数明显不同，表明数据集的划分可能欠妥。

提前终止法可以用于前述的各种训练函数，仅需要将确认样本集的数据传递给训练函数。





但在收敛速度太快的算法（如 `trainlm`）中使用提前终止法要特别谨慎，需要设置训练参数（将系数 `mu` 设置得相对大一些，如设为 1；将 `mu_dec` 和 `mu_inc` 设为接近 1 的数值，如分别设为 0.8 和 1.5），使收敛的速度相对减慢。而对于采用收敛速度较慢算法的训练函数（如 `trainscg` 和 `trainrp`），通常应用提前终止法的效果很好。另外，确认样本集的选择在提前终止法中也很重要，确认样本集对所有训练样本集中的样本而言，应该是富有代表性的。

【例 7-2】 采用提前停止方法提高 BP 网络的推广能力。对于和例 7-13 相同的问题，在本例中我们将采用训练函数 `traingdx` 和提前停止相结合的方法来训练 BP 网络，以提高 BP 网络的推广能力。

在利用提前停止方法时，首先应分别定义训练样本、验证样本或测试样本，其中，验证样本是必不可少的。在本例中，只定义并使用验证样本，即有：

验证样本输入向量：`val.P = [-0.975:.05:0.975]`

验证样本目标向量：`val.T = sin(2*pi*val.P)+0.1*randn(size(val.P))`

值得注意的是，尽管提前停止方法可以和任何一种 BP 网络训练函数一起使用，但是不适合训练速度过快的算法联合使用，比如 `trainlm` 函数，所以本例中采用训练速度相对较慢的变学习速率算法 `traingdx` 函数作为训练函数。

其实现的 MATLAB 代码为：

```
>> clear all
%定义训练样本向量
P = [-1:0.05:1]; %P 为输入向量
randn('seed',78341223); %T 为目标向量
T = sin(2*pi*P)+0.1*randn(size(P));
%绘制训练样本数据点
plot(P,T,'+');
title('训练样本点');
hold on;
plot(P,sin(2*pi*P),'-'); %绘制不含噪声的正弦曲线
title('不含噪声的正弦曲线 ');
%定义验证样本
val.P = [-0.975:0.05:0.975]; %验证样本的输入向量
val.T = sin(2*pi*val.P)+0.1*randn(size(val.P)); %验证样本的目标向量
%创建一个新的前向神经网络
net=newff(minmax(P),[5,1],{'tansig','purelin'},'traingdx');
%设置训练参数
net.trainParam.epochs = 500;
net = init(net);
%训练 BP 网络
[net,tr]=train(net,P,T,[],[],val);
%对 BP 网络进行仿真
A = sim(net,P);
%计算仿真误差
E = T-A;
```

```
MSE=mse(E)
%绘制仿真拟合结果曲线
plot(P,A,P,T,'+',P,sin(2*pi*P),':');
title('仿真拟合曲线')
```

运行程序，得到训练结果如下，训练过程如图 7-2 所示，拟合效果如图 7-3 所示。

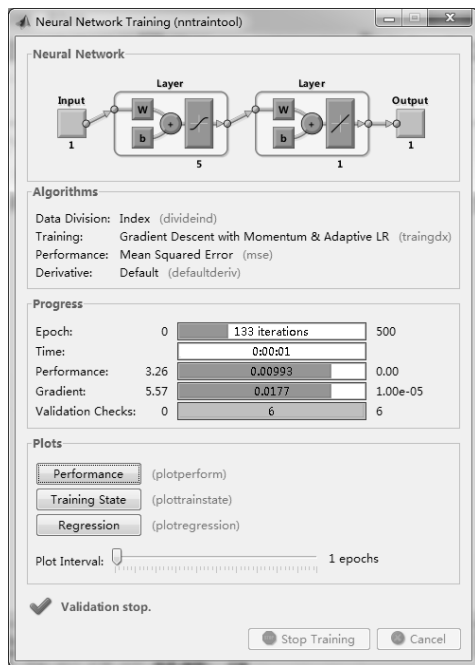


图 7-2 提前终止法训练过程

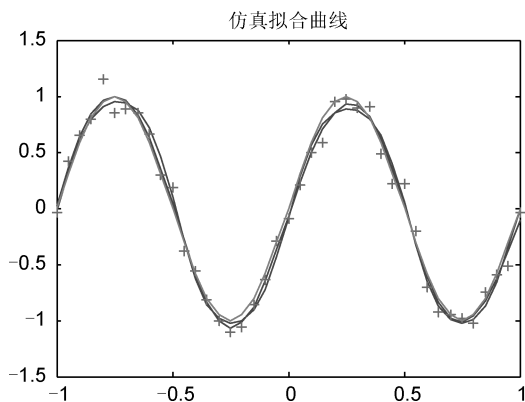


图 7-3 提前终止法数据拟合图

7.3 BP 网络的局限性

BP 神经网络最主要的优点是具有极强的非线性映射能力。理论上，对于一个三层和三层以上的 BP 网络，只要隐层神经元数目足够多，该网络就能以任意精度逼近一个非线性函数。其次，BP 神经网络具有对外界刺激和输入信息进行联想记忆的能力。这是因为它采用了分布并行的信息处理方式，对信息的提取必须采用联想的方式，才能将相关神经元全部调动起来。BP 神经网络通过预先存储信息和学习机制进行自适应训练，可以从不完整的信息和噪声干扰中恢复原始的完整信息。这种能力使其在图像复原、语言处理、模式识别等方面具有重要应用。再次，BP 神经网络对外界输入样本有很强的识别与分类能力。由于它具有强大的非线性处理能力，因此可以较好地非线性分类，解决了神经网络发展史上的非线性分类难题。另外，BP 神经网络具有优化计算能力。BP 神经网络本质上是一个非线性优化问题，它可以在已知的约束条件下寻找一组参数组合，使该组合确定的目标函数达到最小。不过，其优化计算存在局部极小问题，必须通过改进完善。

由于 BP 网络训练中稳定性要求学习效率很小，所以梯度下降法使得训练很慢。动量法因



为学习率的提高通常比单纯的梯度下降法要快一些，但在实际应用中还是速度不够，这两种方法通常只应用于递增训练。

多层神经网络可以应用于线性系统和非线性系统中，对任意函数模拟逼近。当然，感知器和线性神经网络能够解决这类网络问题。但是，虽然理论上是可行的，但实际上 BP 网络并不一定总能求解。

对于非线性系统，选择合适的学习率是一个重要的问题。在线性网络中，学习率过大会导致训练过程不稳定。相反，学习率过小又会造成训练时间过长。和线性网络不同，对于非线性多层网络很难选择很好的学习率。对那些快速训练算法，默认参数值基本上都是最有效的设置。

非线性网络的误差面比线性网络的误差面复杂得多，问题在于多层网络中非线性传递函数有多个局部最优解。寻优的过程与初始点的选择关系很大，初始点如果更靠近局部最优解而不是全局最优解，就不会得到正确的结果，这也是多层网络无法得到最优解的一个原因。为了解决这个问题，在实际训练过程中，应重复选取多个初始点进行训练，以保证训练结果的全局最优性。

网络隐层神经元的数目也对网络有一定的影响。神经元数目太少会造成网络的不适性，而神经元数目太多又会引起网络的过适性。

7.4 BP 网络典型应用

下面通过例子来说明 BP 网络在实际领域中的应用。

7.4.1 用 BP 网络估计胆固醇含量

这是一个将神经网络用于医疗应用的例子。

【例 7-3】 设计一个器械，用于从血样光谱组成的测量中得到血清的胆固醇含量级别，有 261 个病人的血样值，包括 21 种波长谱线的数据，对于这些病人，得到了基于光谱分类的胆固醇含量级别 `hdl,ldl,vldl`。

(1) 样本数据的定义与预处理。

`choles_all.mat` 文件中存储了网络训练所需要的全部样本数据。

利用 `load` 函数可以在工作空间自动载入网络训练所需的输入数据 `p` 和目标数据 `t`，即

```
load choles_all
sizeofp = size(p)
sizeofp = 21264
sizeoft = size(t)
sizeoft = 3264
```

可见，样本集的大小为 264。为了提高神经网络的训练效率，通常要对样本数据进行适当的预处理。首先，利用 `prestd` 函数对样本数据进行归一化处理，使得归一化后的输入和目标数据均服从正态分布，即 `[pn,meanp,stdp,tn,meant,stdt] = prestd(p,t);`



然后,利用 `prepca` 函数对归一化后的样本数据进行主元分析,从而消除样本数据中的冗余成分,起到数据降维的目的。

```
[ptrans,transMat]=prepca(pn,0.001);
[R,Q]=size(ptrans)
R=4;
Q=264
```

可见,主元分析之后的样本数据维数被大大降低,输入数据的维数由 21 变为 4。

(2) 对训练样本、验证样本和测试样本进行划分。

为了提高网络的推广能力和识别能力,训练中采用“提前停止”的方法,因此,在训练之前,需要将上面处理后的样本数据适当划分为训练样本集、验证样本集和测试样本集。

(3) 网络生成与训练。选用两层 BP 网络,其中网络输入维数为 4,输出维数为 3,输出值即为血清胆固醇的三个指标值大小。网络中间层神经元数目预选为 5,传递函数类型选 `tansig` 函数,输出层传递函数选线性函数 `purelin`,训练函数设为 `trainlm`。网络的生成语句如下:

```
net=newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
```

利用 `train` 函数对所生成的神经网络进行训练,训练结果如下:

```
[net,tr]=train(net,ptr,ttr,[],val,test);
```

由图 7-4 可看到,网络训练迭代至第 16 步时提前停止,这是由于验证误差已经开始变大。利用代码可绘制出如图 7-5、图 7-6 及图 7-7 所对应的训练误差、验证误差和测试误差的变化曲线。由图可见,验证误差和测试误差的变化趋势基本一致,说明样本集的划分基本合理。由训练误差曲线可见,训练误差结果也是比较满意的。

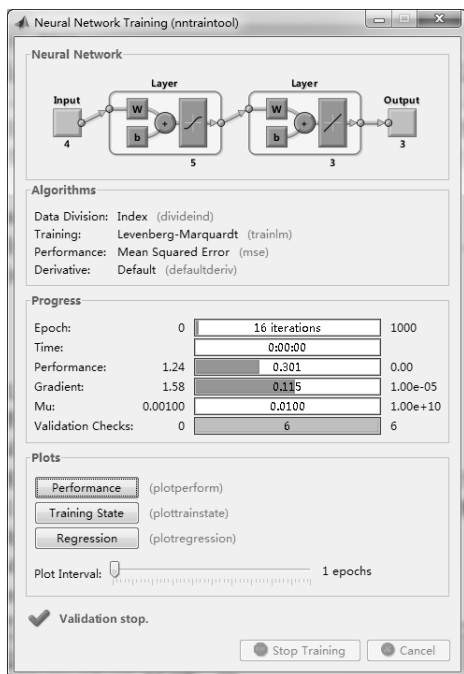


图 7-4 网络训练过程

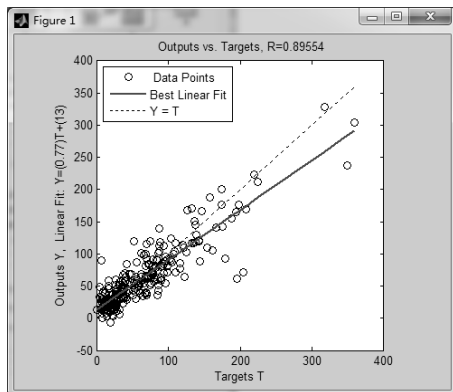
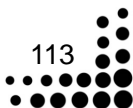


图 7-5 训练误差曲线



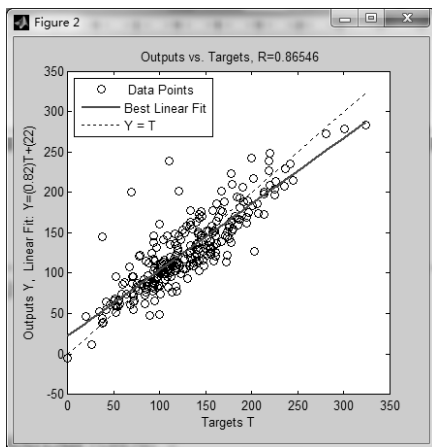


图 7-6 验证误差曲线

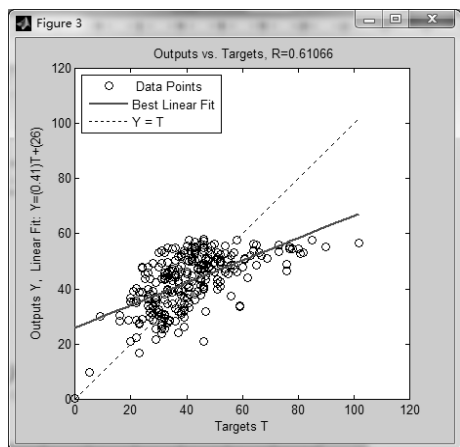


图 7-7 测试误差曲线

(4) 网络仿真。为了进一步检验训练后网络的性能，下面对训练结果作进一步仿真分析。利用 `postreg` 函数可以对网络仿真的输出结果和目标输出作线性回归分析，并得到两者的相关系数，从而可以作为网络训练结果优劣的判别依据。仿真与线性回归分析如下：

```
an = sim(net,ptrans);
a = poststd(an,meant,stdt);
for i=1:3
    figure(i)
    [m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end
```

完成的 MATLAB 代码为：

```
>>%导入原始测量数据
load choles_all;
%对原始数据进行规范化处理，prestd 是对输入数据和输出数据进行规范化处理，
%prepca 可以删除一些数据，适当地保留了变化不小于 0.01 的数据
[pn,meanp,stdp,tn,meant,stdt]=prestd(p,t);
[ptrans,transMat]=prepca(pn,0.001);
[R,Q]=size(ptrans)
%将原始数据分成几个部分作为不同用途，1/4 用于确证，1/4 用于测试，1/2 用于训练网络
iitst=2:4:Q;
iiival=4:4:Q;
iitr=[1:4:Q 3:4:Q];
%vv 是确证向量，.P 是输入，.T 是输出，vt 是测试向量
vv.P=ptrans(:,iiival);
vv.T=tn(:,iiival);
vt.P=ptrans(:,iitst);
vt.T=tn(:,iitst);
```



```

ptr=ptrans(:,iitr);
ttr=tn(:,iitr);
%建立网络，隐层中设计5个神经元，由于需要得到的是3个目标，所以网络需要有3个输出
net=newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
%训练网络
net.trainParam.show=5;
[net,tr]=train(net,ptr,ttr,[],[],vv,vt);
%绘出训练过程中各误差的变化曲线
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,'g',tr.epoch,tr.tperf,'-b');
legend('训练','确证','测试',-1);
ylabel('平方误差');
xlabel('时间');
pause;
%将所有数据通过网络（包括训练，确证，测试），然后得到网络输出和相应目标进行线性回归，
%对网络输出进行反规范化变换，并绘出个各级别的线性回归结果曲线
an=sim(net,ptrans);
a=poststd(an,meant,stdt);
%得到3组输出，所以进行3次线性回归
for i=1:3
figure(i)
[m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end

```

网络输出数据和目标数据线性回归后，前面两个输出对目标的跟踪比较好，相应的 R 值接近 0.9，而第三个输出却并不理想。

如果把隐层数目改为 20 个，网络训练的 3 种误差非常接近，得到的结果 R 也相应提高，但不代表神经元越多就越精确。

多层神经网络能够对任意的线性或者非线性函数进行逼近，其精度也是任意的。但是 BP 网络不一定能找到解。训练时，学习速率太快可能引起不稳定，太慢则要花费太多时间，不同的训练算法也对网络的性能有很大影响。BP 网络对隐层的神经元数目也是很敏感的，太少则很难适应，太多则可能设计出超适应网络。

7.4.2 线性神经网络在信号预测中的应用

【例 7-4】 实现自适应预测的线性网络。设自适应滤波器如图 7-8 所示，该滤波器的目的是要从输入信号的前两个时刻的值预测当前时刻的值。

图中 D 为延迟单元，多个延迟单元可以构成抽头延迟线，如图 7-9 所示。

设输入信号为一随机序列，试编写 MATLAB 程序，画出上述自适应滤波器的输入、输出波形。

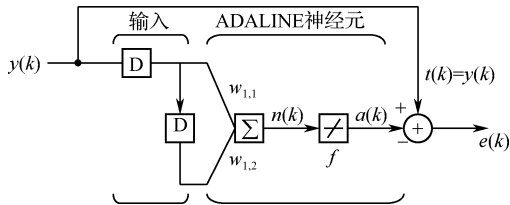


图 7-8 自适应滤波器示意图

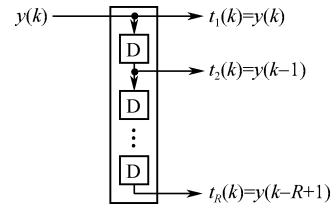


图 7-9 抽头延迟线

其实现的 MATLAB 程序代码如下：

```
>>clear all;
%定义输入向量和目标向量
time=0.5:0.5:20;           %时间变量
y=(rand(1,40)-0.5)*4;      %定义随机输入信号
p=con2seq(y);               %将随机输入向量转换为串行向量
delays=[1 2];               %定义 ADALINE 神经元输入延迟量
t=p;                         %定义 ADALINE 神经元的数目向量
%创建线性神经网络
net=newlin(minmax(y),1,delays,0.0005);
%线性神经网络的自适应调整（训练）
net.adaptParam.passes=70;
[net,a,output]=adapt(net,p,t); %输出信号 output 为网络调整过程中的误差
%绘制随机输入信号\输出信号的波形
hold on;
subplot(3,1,1);plot(time,y,'r*-'); %输出信号 output 为网络调整过程中的误差
xlabel('t','position',[20.5,-1.8]);
ylabel('随机输入信号 s(t)');
axis([0 20 -2 2]);
subplot(3,1,2);
output=seq2con(output);
plot(time,output{1},'ko-'); %绘制预测输出信号的波形
xlabel('t','position',[20.5, -1.8]);
ylabel('预测输出信号 y(t)');
axis([0 20 -2 2]);
subplot(3,1,3);
e=output{1}-y;
plot(time,e,'k-'); %绘制误差曲线
xlabel('t','position',[20.5, -1.8]);
ylabel('误差曲线 e(t)');
axis([0 20 -2 2]);
hold off;
```

运行程序，效果如图 7-10 所示。

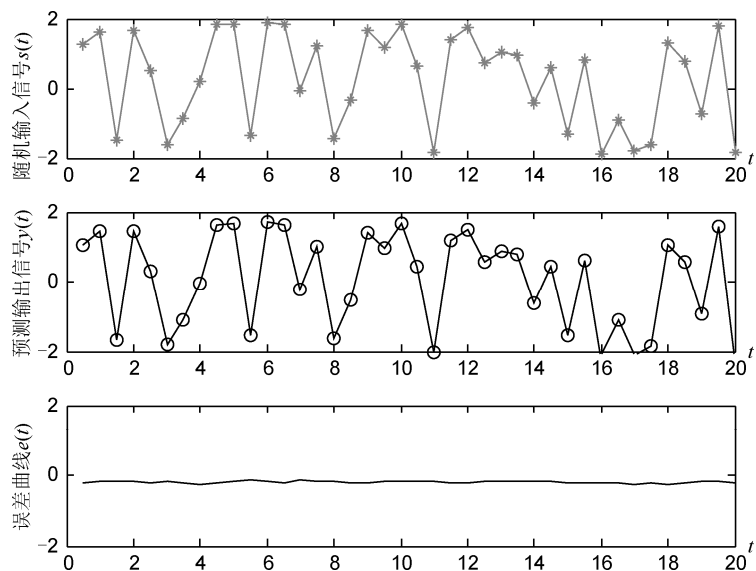


图 7-10 线性神经网络信号预测效果

从图中可以看出，输出信号波形与输入信号波形基本一致，误差较小，输出波形较好地预测了输入波形。

值得一提的是，在程序设计中，需要注意学习率和训练步长的选择。学习率过大，学习的过程将不稳定，且误差会更大；学习率过小，学习的过程将变慢，需要的训练步长数将加大。选择不当，将得不到满意的结果。

就线性神经网络本身而言，它与感知器一样，只能解决线性可分的模式分类，但 LMS 算法比感知器的 δ 学习算法更有效，因为它使均方误差最小，可以使各分类模式远离判决边界，从而使网络具有更好的抗噪性能。另一方面，ADALINE 网络至今仍然广泛应用于各种实际系统中，特别是在自适应滤波方面，用途更为广泛。

【例 7-5】 对给定的输入样本数据及期望数据进行预测。

```
>> %原始数据输入
p=[2845 2833 4488;2833 4488 4554;4488 4554 2928;4554 2928 3497;2928 3497 2261;...
3497 2261 6921;2261 6921 1391;6921 1391 3580;1391 3580 4451;3580 4451 2636;...
4451 2636 3471;2636 3471 3854;3471 3854 3556;3854 3556 2659;3556 2659 4335;...
2659 4335 2882;4335 2882 4084;4335 2882 1999;2882 1999 2889;1999 2889 2175;...
2889 2175 2510;2175 2510 3409;2510 3409 3729;3409 3729 3489;3729 3489 3172;...
3489 3172 4568;3172 4568 4015;]';
%期望输出
t=[4554 2928 3497 2261 6921 1391 3580 4451 2636 3471 3854 3556 2659 ...
4335 2882 4084 1999 2889 2175 2510 3409 3729 3489 3172 4568 4015 3666];
%目标数据
ptest=[2845 2833 4488;2833 4488 4554;4488 4554 2928;4554 2928 3497;2928 3497 ...
2261;3497 2261 6921;2261 6921 1391;6921 1391 3580;1391 3580 4451;3580 4451 2636;...
4451 2636 3471;2636 3471 3854;3471 3854 3556;3854 3556 2659;3556 2659 4335;...
```



```

2659 4335 2882;4335 2882 4084;4335 2882 1999;2882 1999 2889;1999 2889 2175;...
2889 2175 2510;2175 2510 3409;2510 3409 3729;3409 3729 3489;3729 3489 3172;...
3489 3172 4568;3172 4568 4015;4568 4015 3666];
[pn,minp,maxp,tn,mint,maxt]=premnmx(p,t); %将数据归一化
NodeNum1=20; %隐层第一层节点数
NodeNum2=40; %隐层第二层节点数
TypeNum=1; %输出维数
TF1='tansig';
TF2='tansig';
TF3='tansig';
net=newff(minmax(pn),[NodeNum1,NodeNum2,TypeNum],{TF1 TF2 TF3},'traingdx');
%网络创建 traingdm
net.trainParam.show=50;
net.trainParam.epochs=50000; %训练次数设置
net.trainParam.goal=1e-5; %训练所要达到的精度
net.trainParam.lr=0.01; %学习速率
net=train(net,pn,tn);
p2n=trmnmx(pst,minp,maxp); %测试数据的归一化
an=sim(net,p2n);
[a]=postmnmx(an,mint,maxt) %数据的反归一化，即最终想得到的预测结果
plot(1:length(t),t,'o',1:length(t)+1,a,'+');
title('o 表示预测值--- *表示实际值')
grid on
m=length(a); %向量 a 的长度
t1=[t,a(m)];
error=t1-a; %误差向量
figure
plot(1:length(error),error,'-.')
title('误差变化图')
grid on

```

运行程序，得到训练过程如图 7-11 所示。

得到的实际与预测值散点图及误差变化曲线如图 7-12 及图 7-13 所示。预测值为：

```

a =
    1.0e+03 *
Columns 1 through 8
    4.5537    2.9280    3.4965    2.2601    6.8764    1.3984    3.5800    4.4510
Columns 9 through 18
    2.6357    3.4717    3.8543    3.5571    2.6607    4.3350    2.8816    4.0844
Columns 19 through 24
    2.0000    2.8893    2.1755    2.5112    3.4086    3.7290    3.4885    3.1696
Columns 15 through 28

```

4.5689 4.0152 3.6658 2.1525

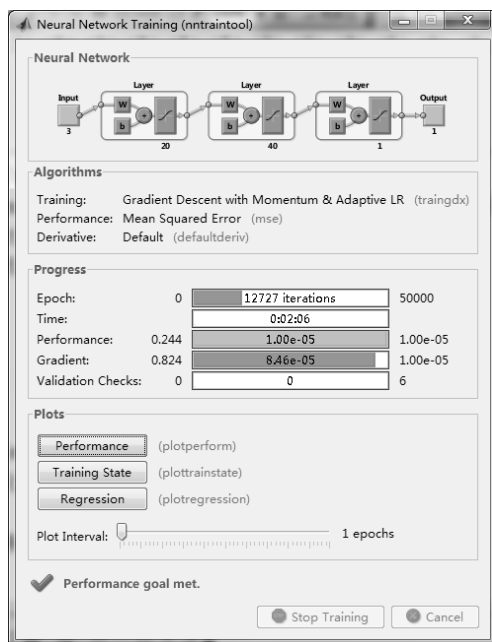


图 7-11 BP 网络训练过程图

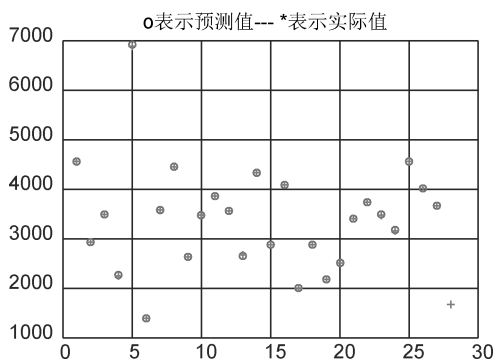


图 7-12 实际与预测值散点图

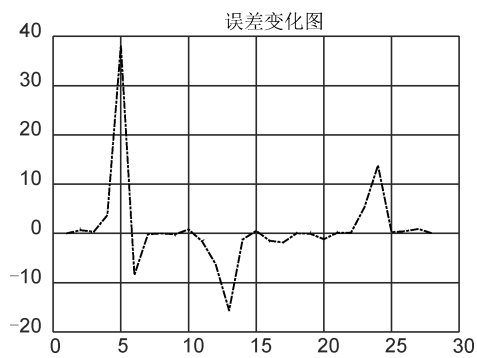


图 7-13 误差变化曲线

第 8 章 线性神经网络算法分析与实现

线性神经网络是一种简单的神经元网络，它可以由一个或多个线性神经元构成。1963 年由美国斯坦福大学教授 Berhard Widrow 提出的自适应线性元件网络(Adaptive Linear Element, Adaline)是线性神经网络最早的典型代表，它是一个由输入层和输出层构成的单层前馈型网络。它与感知器神经网络的不同之处在于其每个神经元的传输函数为线性函数，因此自适应线性神经网络的输出可以取任意值，而感知器神经网络的输出只能是 1 或 0。线性神经网络采用由 Berhard Widrow 和 Marcian Hoff 共同提出的一种新的学习规则，也称为 Widrow-Hoff 学习规则，或者 LMS (Least Mean Square) 算法来调整网络的权值和阈值。自适应线性神经网络的学习算法比感知器网络的学习算法的收敛速度和精度都有较大的提高。自适应线性神经网络主要用于函数逼近、信号预测、系统辨识、模式识别和控制等领域。

8.1 线性神经网络工具箱函数

在 MATLAB 神经网络工具箱中提供了大量的与线性神经网络相关的函数。在 MATLAB 工作空间的命令行中输入“help linnet”，便可得到与线性网络相关的信息，进一步利用 help 命令又可得到相关函数的详细介绍。下面分别对这些线性函数进行介绍。

8.1.1 创建函数

在 MATLAB 神经网络工具箱中提供了 newlin 函数及 newlind 函数，用于创建线性神经网络。

1. newlin 函数

该函数可以创建一个线性层。所谓线性层是一个单独的层次，它的权函数为 dotprod，输入函数为 netsum，传递函数为 purelin。线性层一般做信号处理和预测中的自适应滤波器，其调用格式为：

```
net=newlin(PR,S,ID,LR)
```

其中，

PR 是由 R 个输入元素的最大值和最小值组成的 $R \times 2$ 维矩阵；

S 是输入向量的数目；

ID 是输入延迟向量，默认为[0]；



LR 是学习速率，默认为 0.01；

net 是函数返回值，一个新的线性层。

net=newlin 表示在一个对话框中创建一个新的网络。

注意：如果用 0 代替参数 ID，用输入向量的矩阵 P 代替参数 LR，那么函数 newlin(PR,S,0,P) 返回的线性层的稳定学习速率对于 P 来说是最大的。

【例 8-1】应用 newlin 设计一个双输入单输出线性神经网络，输入向量范围是[-1 1; -1 1]，学习率为 1。

```
>>clear all;
net=newlin([-1 1; -1 1],1);
```

此时网络权值和阈值默认为 0，

```
W=net.IW{1,1}
W =
    0    0
b=net.b{1}
b =
    0
```

当然，可以给权值和阈值赋值，如：

```
net.IW{1,1}=[2 3];
W=net.IW{1,1}
W =
    2    3
net.b{1}=[-4];
b=net.b{1}
b =
   -4
```

下面，对于输入向量 p 应用函数 sim 进行仿真计算，计算出相应的函数输出 a。

```
p=[5;6];
a=sim(net,p)
a =
   24
```

可见，应用 newlin 函数可以构建一个线性神经网络并随意调整权值和阈值，还可以利用 sim 函数进行仿真。

2. newlind 函数

该函数可以设计一个线性层，它通过输入向量和目标向量来计算线性层的权值和阈值，其调用格式为：

```
net = newlind(P,T,Pi)
```

其中，



P 是 Q 组输入向量组成的 $R \times Q$ 维矩阵;

T 是 Q 组目标分类向量组成的 $S \times Q$ 维矩阵;

P_i 是初始输入延迟状态的 ID 个单元阵列, 每个元素 $P_i\{i,k\}$ 都是一个 $R_i \times Q$ 维的矩阵, 默认为空。

`net` 是函数返回值, 一个线性层, 它的输出误差平方和对于输入 P 来说具有最小值。

【例 8-2】 利用线性神经网络进行自适应预测。

```
>> clear all;
T1=0:0.24:4;
T=sin(T1*5*pi);
Q=length(T);           %生成一个信号作为预测信号
%将信号 T 延时 1~5 个时间步长得到的网络的输入 X
X=zeros(5,Q);
X(1,2:Q)=T(1,1:(Q-1));
X(2,3:Q)=T(1,1:(Q-2));
X(3,4:Q)=T(1,1:(Q-3));
X(4,5:Q)=T(1,1:(Q-4));
X(5,6:Q)=T(1,1:(Q-5));
figure;plot(T1,T);      %绘制信号 T 的曲线
hold on;
net=newlind(X,T);       %构建一个线性层
y=sim(net,X);           %网络仿真
plot(T1,y,'.',T1,T-y,'*');
legend('网络待预测目标信号','网络预测输出','网络预测误差');
```

运行程序, 效果如图 8-1 所示。

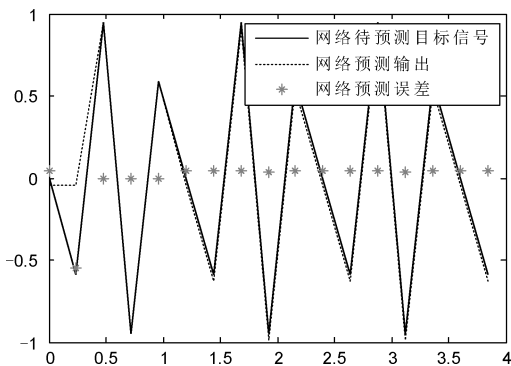


图 8-1 网络的自适应预测曲线

8.1.2 学习函数

在 MATLAB 神经网络工具箱中提供了 `learnwh` 函数及 `maxlinlr` 函数, 用于实现线性神经网络的学习。



1. learnwh 函数

该函数为 Widrow-Hoff 学习函数，也称为 delta 准则或最小方差准则学习函数。它可以修改神经元的权值和阈值，使输出误差的平方和最小。它沿着误差平方和的下降最快方向连续调整网络的权值和阈值，由于线性网络的误差性能表面是抛物面，仅有一个最小值，因此可以保证网络是收敛的，前提是学习率不超出由函数 `maxlinlr` 计算得到的最大值，其调用格式为：

```
[dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
```

其中，

W 为 $S \times R$ 维的加权矩阵（或为 $S \times 1$ 的阈值矩阵）。

P 为 Q 组 R 维的输入向量（或为 Q 个单值输入）。

Z 为 Q 组 S 维的加权输入向量。

N 为 Q 组 S 维的网络输入向量。

A 为 Q 组 S 维的网络输出向量。

T 为 Q 组 S 维的目标向量。

E 为 Q 组 S 维的误差向量。

gW 为 $S \times R$ 维的性能参数的梯度。

gA 为 Q 组 S 维的性能参数的输出梯度。

D 为神经元距离。

LP 为学习参数，如果没有则为空。

LS 为学习状态，初始值为空。

dW 为 $S \times R$ 维权值（或阈值）的变化矩阵。

LS 为新的学习状态。

`info = learnwh('code')` 为针对不同的 `code` 返回相应的有用信息，包括：

- 当 `code=pnames` 时表示返回学习参数的名称；
- 当 `code=pdefaults` 时表示返回默认的学习参数；
- 当 `code=needg` 时表示如果函数使用了 gW 或 gA ，则返回 1。

【例 8-3】 对给定的数据进行 Widrow-Hoff 学习。

```
>> clear all;
p = rand(2,1);
e = rand(3,1);
lp.lr = 0.5;
dW = learnwh([],p,[],[],[],[],e,[],[],lp,[])
```

运行程序，输出如下：

```
dW =
    0.2356    0.2323
    0.3884    0.3830
    0.0689    0.0679
```



2. maxlinlr 函数

该函数为分析函数，用于计算线性层的最大学习速率，其调用格式为：

```
lr = maxlinlr(P)
lr = maxlinlr(P,'bias')
```

其中，

P 为输入向量的 $R \times Q$ 维矩阵。

lr=maxlinlr(P)为针对不带阈值的线性层得到一个所需要的最大学习速率。

lr=maxlinlr(P,'bias')针对带有阈值的线性层得到一个所需要的最大学习速率。

【例 8-4】 在给定输入 P 的情况下，分“带阈值”与“不带阈值”两种情况求得该线性层所需的最大学习速率。

```
>> clear all;
P=[1 2 -4 7; 0.1 3 10 6];
disp('不带阈值速率为: ')
lr1 = maxlinlr(P)
disp('带阈值速率为: ')
lr2 = maxlinlr(P,'bias')
```

运行程序，输出如下：

不带阈值速率为：

```
lr1 =
    0.0069
```

带阈值速率为：

```
lr2 =
    0.0067
```

8.1.3 性能函数

在 MATLAB 神经网络工具箱中提供了 purelin 函数，实现线性神经网络的性能检测。

purelin 函数实现线性神经网络的传输。神经元最简单的传输函数是简单地从神经元输入到输出的线性传输函数，输出仅仅被神经元所附加的偏差所修正。线性传输函数常用于由 Widrow-Hoff 或 BP 准则来训练的神经网络中。purelin 函数的调用格式为：

$A = \text{purelin}(N, FP)$ ，N 为 $S \times Q$ 维的网络输入（列）向量矩阵，FP 为性能参数（可忽略），返回网络输入向量 N 的输出矩阵 A。

$dA_dN = \text{purelin}('dn', N, A, FP)$ ，返回 A 关于 N 的导数 dA_dN ，如果 A 或 FP 没有给出或为空矩阵，则 FP 返回默认参数。

info = purelin('code')，依据 code 值的不同，返回不同的信息，包括：

- 当 code=name 时表示返回传输函数的全称。



- 当 code=active 时返回传输函数最小、最大值的有效输入范围。
- 当 code=output 时返回传输函数的最小、最大值的二元适量。
- 当 code=fullderiv 时，根据 dA_dN 是 $S \times S \times Q$ 还是 $S \times Q$ 来确定返回 1 还是 0。
- 当 code=fpnames 时返回函数参数的名称。
- 当 code=fpdefaults 时返回默认的函数参数。

【例 8-5】产生一个线性 S 型传递函数。

```
>> clear all;
n = -5:0.1:5;
a = purelin(n);
plot(n,a)
```

运行程序，效果如图 8-2 所示。

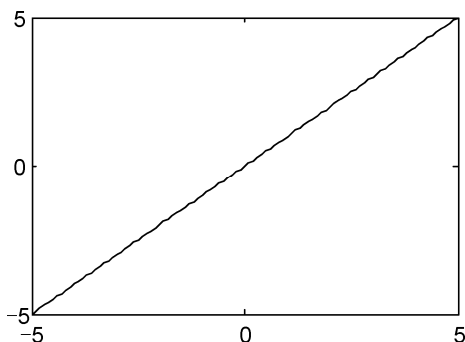


图 8-2 S 型传递函数

8.2 线性神经网络模型及结构

1. 线性神经网络模型

一个线性的具有 r 个输入节点的自适应线性神经网络模型如图 8-3 所示。这个神经网络有一个线性激活函数，被称为 Adaline。和感知器一样，偏差可以用来作为网络的一个可调参数，提供额外可调的自由变量以获得期望的网络特性。线性神经网络可以训练学习一个与之对应的输入/输出的函数关系，或线性逼近任意一个非线性函数，但它不能产生任何非线性的计算特性。

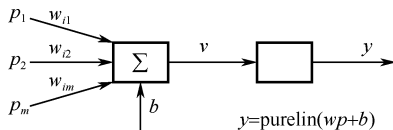


图 8-3 Adaline 神经网络

当自适应线性网络由 s 个神经元相关联形成一层网络时，此自适应线性神经网络又称为 Madaline，如图 8-4 所示。

2. 线性神经网络的结构

单层线性神经网络的结构如图 8-5 所示，其中网络具有 R 个输入， S 个神经元，并通过

权值 w_{ij} 连接，下标 i 和 j 表示权值 w_{ij} 连接着第 j 个神经元和第 i 个输入，通过网络后产生 S 个输出。

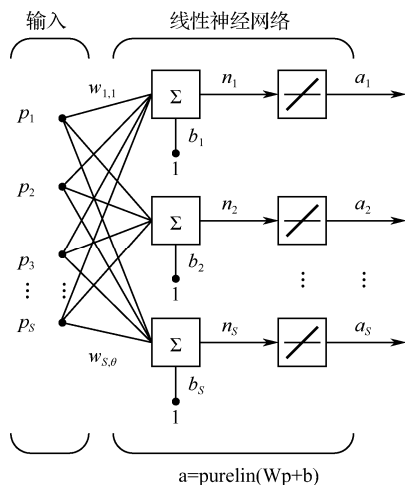


图 8-4 Madaline 神经网络

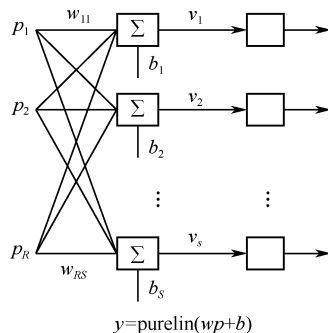


图 8-5 线性神经网络结构

8.3 线性神经网络的学习算法与训练

8.3.1 线性神经网络的学习算法

W-H 学习规则是由威德罗和霍夫提出的用来修正权向量的学习规则，所以用他们两人姓氏的第一个字母来命名。W-H 学习规则可以用来训练一层网络的权值和偏差，使之线性地逼近一个函数式而进行模式联想（Pattern Association）。

定义一个线性网络的输出误差函数为：

$$E(W, B) = \frac{1}{2} [T - A]^2 = \frac{1}{2} [T - WP - B]^2 \quad (8-1)$$

由式（8-1）可以看出：线性网络具有抛物线型误差函数所形成的误差表面，所以只有一个误差最小值。通过 W-H 学习规则来计算权值和偏差的变化，并使网络误差的平方和最小化，总能够训练一个网络的误差趋于这个最小值。另外，很显然， $E(W, B)$ 只取决于网络的权值及目标向量。我们的目的是通过调节权向量，使 $E(W, B)$ 达到最小值。所以在给定 $E(W, B)$ 后，利用 W-H 学习规则修正权向量和偏差向量，使 $E(W, B)$ 从误差空间的某一点开始，沿着 $E(W, B)$ 的斜面向下滑行。根据梯度下降法，权向量的修正值正比于当前位置上 $E(W, B)$ 的梯度，对于第 i 个输出节点有：

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta [t_i - a_i] p_j \quad (8-2)$$

或表示为：



$$\begin{aligned}\Delta w_{ij} &= \eta \delta_i p_j \\ \Delta b_i &= \eta \delta_i\end{aligned}\quad (8-3)$$

这里 δ_i 定义为第 i 个输出节点的误差:

$$\delta_i = t_i - a_i \quad (8-4)$$

式 (8-3) 称为 **W-H** 学习规则, 又叫 δ 规则, 或称为最小均方差算法 (LMS)。**W-H** 学习规则的权值变化量正比于网络的输出误差及网络的输入向量。它不需求导数, 所以算法简单, 又具有收敛速度快和精度高的优点。

式 (8-3) 中的 η 为学习速率。在一般的实际运用中, 实践表明, η 通常取一接近 1 的数, 或取值为

$$\eta = 0.99 \times \frac{1}{\max[\det(PP^T)]} \quad (8-5)$$

这样的选择可以达到既快速又正确的结果。

学习速率的这一取法在神经网络工具箱中用函数 `maxlinlr` 来实现。式 (8-5) 可实现为:

$$lr = 0.99 \maxlinlr(P)$$

其中 `lr` 为学习速率。

W-H 学习规则的函数用 `learnwh` 来实现, 另外, 加上线性自适应网络输出函数 `purelin`, 可以写出 **W-H** 学习规则的计算程序为:

```
A=purelin(W*P+B);
E=T-A;
dW=learnwh(P,[],[],[],E,[],[],lr,[]);
dB=learnwh(B,ones(1,Q),[],[],[],E,[],[],lr,[]);
W=W+dW;
B=B+dB
```

采用 **W-H** 规则训练自适应线性元件使其能够得以收敛的必要条件是被训练的输入向量必须是线性独立的, 且应适当地选择学习速率以防止产生振荡现象。

W-H 学习规则的步骤如下所述。

第一步: 设置变量和参量。

$X(n) = [1, x_1(n), x_2(n), \dots, x_m(n)]^T$ 为输入向量, 或称训练样本。

$W(n) = [b(n), w_1(n), w_2(n), \dots, w_m(n)]^T$ 为权值向量。

$b(n)$ 为偏差。

$y(n)$ 为实际输出。

$d(n)$ 为期望输出。

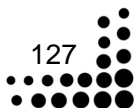
η 为学习速率。

n 为迭代次数。

第二步: 初始化, 赋给 $W_j(0)$ 一个较小的随机非零值, $n=0$ 。

第三步: 对于一组输入样本 $X(n) = [1, x_1(n), x_2(n), \dots, x_m(n)]^T$ 和对应的期望输出 d , 计算:

$$\begin{aligned}e(n) &= d(n) - X^T(n)W(n) \\ W(n+1) &= W(n) + \eta X^T(n)e(n)\end{aligned}$$



第四步：判断是否满足条件，若满足，算法结束；若不满足，将 n 值增加 1，转到第三步重新执行。

注意：在以上学习算法的第四步需要判断是否满足条件，这里的条件可以是误差小于设定的值 ε ，即 $|e(n)| < \varepsilon$ ；或者是权值的变化已很小，即 $|w(n+1) - w(n)| < \varepsilon$ 。另外，在实现过程中还应设定最大的迭代次数，以防止算法万一不收敛时，程序进入死循环。

8.3.2 线性神经网络的训练

自适应线性元件的网络训练过程可以归纳为以下三个步骤。

(1) 表达：计算训练的输出向量 $A=WP+B$ ，以及与期望输出之间的误差 $E=T-A$ 。

(2) 检查：将网络输出误差的平方和与期望误差相比较，如果其值小于期望误差，或训练已达到事先设定的最大训练次数，则停止训练，否则继续。

(3) 学习：采用 $W-H$ 学习规则计算新的权值和偏差，并返回到 (1)。

每进行一次上述三个步骤，被认为是完成一个训练循环次数。

如果网络训练获得成功，那么当一个不在训练中的输入向量输入到网络中时，网络趋于产生一个与其相联想的输出向量。这个特性被称为泛化，这在函数逼近及输入向量分类的应用中是相当有用的。

如果经过训练，网络仍不能达到期望目标，可以有两种选择：或检查一下所要解决的问题，是否适用于线性网络，或对网络进行进一步的训练。

虽然只适用于线性网络， $W-H$ 学习规则仍然是重要的，因为它展现了梯度下降法是如何训练一个网络的，此概念后来发展成反向传播法，使之可以训练多层非线性网络。

【例 8-6】 设计一个双输入单输出线性神经网络，输入向量范围为 $[-1 \ 1; -1 \ 1]$ ，学习率为 1，网络结构如图 8-6 所示。

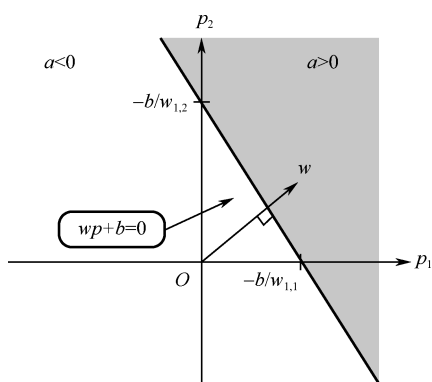


图 8-6 二输入线性神经网络分类示意图

对于下列样本应用线性滤波器进行分类。四组数
组分别包括四个输入向量和相应的期望值。

$$\left\{ p_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, t_1 = 0 \right\}, \left\{ p_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_2 = 1 \right\}$$

$$\left\{ p_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, t_3 = 0 \right\}, \left\{ p_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, t_4 = 1 \right\}$$

其实现的 MATLAB 代码为：

```
>> clear all;
P=[2 1 -2 -1;2 -2 2 1];
T=[0 1 0 1];
%利用 train 函数训练网络，新网络初始权值和阈值默认为 0，设置训练精度为 0.1
```




```
net=newlin([-2 2; -2 2],1);
net.trainParam.goal=0.1;
[net,tr]=train(net,P,T);
disp('经过训练后的权值: ')
w=net.iw{1,1}
disp('经过训练后的阈值: ')
b=net.b{1}
disp('仿真验证: ')
y=sim(net,P)
disp('显示计算误差: ')
err=T-sim(net,P)
```

运行程序，输出如下，效果如图 8-7 所示。

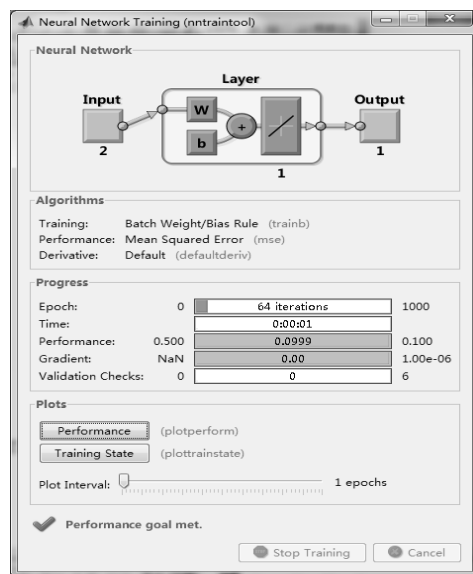


图 8-7 网络训练效果图

经过训练后的权值:

```
w =
    -0.0615    -0.2194
```

经过训练后的阈值:

```
b =
    0.5899
```

仿真验证:

```
y =
    0.0282    0.9672    0.2741    0.4320
```

显示计算误差:

```
err =
    -0.0282    0.0328   -0.2741    0.5680
```

由以上结果可看出，结果并没有达到训练精度。如果提高精度，延迟训练时间，应该能够达到满意的结果，该问题不可能实现零误差，这是线性网络的局限性造成的。

8.4 线性神经网络的滤波器

首先需要了解一下应用于线性神经网络中的触发延迟线，如图 8-8 所示。

从左端接入输入向量，通过 TDL，发生 $N-1$ 延迟。TDL 的输出是一个 N 维向量，相当于当前输入向量的前一时间段的输入信号。

若在线性神经网络中应用了触发延迟线，则将产生如图 8-9 所示的线性滤波器。

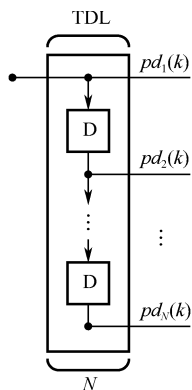


图 8-8 触发延迟线

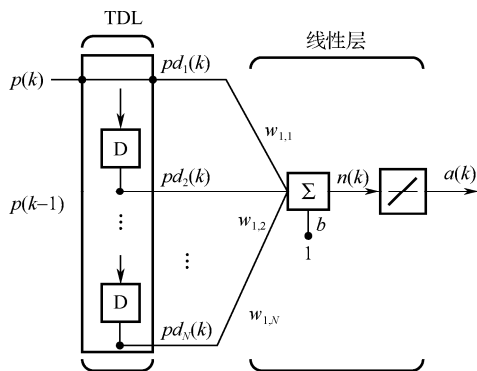


图 8-9 线性滤波器

滤波器的输出如下：

$$a(k) = \text{purelin}(wp + b) = \sum_{i=1}^R w_{1,i} a(k-i+1) + b \quad (8-6)$$

自适应滤波器网络常用于数字信号的处理领域中。

在 MATLAB 中，TDL 网络可以通过以下两种方式实现。第一种是通过原网络输入 P 根据需要构造出延时网络输入 P_d ，然后按无延时的网络设计方法进行网络设计。将输入 $P_{1 \times Q}$ 扩展成 $P_{dN \times Q}$ ($Q \geq N$) 的方法为：如图 8-9 所示的 TDL 结构，TDL 从上到下各个延时单元的初始输出为 $P[0], P[-1], \dots, P[2-N]$ ，而输入为 $P = \{P[1], P[2], \dots, P[Q]\}$ ，扩展成 P_d 的结果为：

$$P_d = \begin{bmatrix} P[1] & P[2] & \cdots & P[Q] \\ P[0] & P[1] & \cdots & P[Q-1] \\ \vdots & \vdots & \ddots & \vdots \\ P[-N] & P[1-N] & \cdots & P[Q-N-1] \end{bmatrix}$$

值得注意的是，含有延时的输入向量 P_d 是行向量。另外， P_d 为原网络输入从左到右所对应的 TDL 从上到下的输出，所以 P_d 中的顺序是从左到右递增的。

第二种方法是首先确定自适应线性网络 newlin 函数中的延时参数 N ，然后给定以下延时



量的赋值:

```
net=newlin(minmax(P),S);
net.inputWeights{1,1}.delays=[0,1,2,...,N-1];
```

或直接写进创建函数中:

```
net=newlin(minmax(P),S,[0,1,2,...,N-1]);
```

下面给出一个带有延时的自适应线性网络的建立及其仿真。

【例 8-7】 假定一个输入按输入顺序排列,其数值分别为 3、4、5 和 6。该输入量自身产生两次延时,其延时量的初始输入值分别为 1 和 2。用一个自适应线性网络为其滤波所获得的权矩阵为[7 8 9]。试计算在给定的输入值下该网络的输出值。

```
>> clear all;
P={3 4 5 6};
net=newlin(minmax(P),1);           %创建一个线性网络
net.inputWeights{1,1}.delays=[0 1 2]; %赋予网络的延时矩阵
%赋给网络权值和偏差
net.IW{1,1}=[7 8 9];
net.b{1}=[0];
%赋给延时环节初始值
Pi={1 2};
%对网络进行仿真
[a,pf]=sim(net,P,Pi);
disp('输出顺序结果为: ')
a
disp('延时输出的最后值为: ')
pf
%给定输出的顺序值
T={10 20 30 40};
net.adaptParam.passes=10;           %给定网络的循环次数
[net,y,e,pf,af,tr]=adapt(net,P,T,Pi)
wts=net.iw{1,1}                     %返回最终的权值
bias=net.b{1}                       %返回最终的偏差
```

运行程序,输出如下:

输出顺序结果为:

```
a =
    [46]    [70]    [94]    [118]
```

延时输出的最后值为:

```
pf =
    [5]    [6]
y =
    [46]    [62.4400]    [67.7284]    [61.9799]
```

```
e =  
    [-36]    [-42.4400]    [-37.7284]    [-21.9799]  
pf =  
    [5]    [6]  
af =  
    Empty cell array: 1-by-0  
tr =  
    timesteps: [1 2 3 4]  
    perf: [1296 1.8012e+003 1.4234e+003 483.1164]  
wts =  
    1.0172    3.3987    5.7802  
bias =  
    -1.3815
```

如果让网络训练更多的次数，输出结果将更接近期望值。

第9章 感知器网络算法分析与实现

9.1 单层感知器

1958年，美国心理学家 Frank Rosenblatt 提出一种具有单层计算单元的神经网络，称为 Perceptron，即感知器。感知器模拟人的视觉接收环境信息，并由神经冲动进行信息传递。感知器研究中首次提出了自组织、自学习的思想，而且对所能解决的问题存在着收敛算法，并能从数学上严格证明，因而对神经网络的研究起了重要推动作用。

单层感知器的结构与功能都非常简单，以至于目前在解决实际问题时很少被采用，但它在神经网络研究中具有重要意义，是研究其他网络的基础，而且较易学习和理解，适合于作为学习神经网络的起点。

9.1.1 单层感知器模型

单层感知器是一个具有一层神经元、采用阈值激活函数的前向网络，通过对网络权值的训练，可以使感知器对一组输入向量的响应达到元素为 0 或 1 的目标输出，从而实现对输入向量分类的目的。图 9-1 是单层感知器神经元模型图。

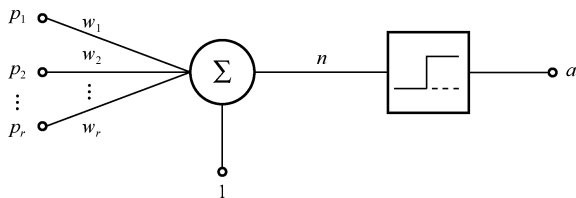


图 9-1 单层感知器神经元模型

图 9-1 中，每一个输入分量 $p_j (j=1,2,\dots,r)$ 通过一个权值分量 w_j 进行加权求和，并作为阈值函数的输入。偏差 b 的加入使得网络多了一个可调参数，为网络输出达到期望的目标向量提供了方便。感知器特别适合解决简单的模式分类问题。

感知器实际上是在 MP 模型的基础上加上学习功能，使其权值可以调节的产物。罗森布拉特研究了单层及具有一个隐含层的感知器。但在当时只能证明单层感知器可以将线性分为输入向量进行正确划分，所以在此所说的感知器是指单层感知器。多层网络因为要用到后面将要介绍的反向传播法进行权值修正，所以把它们均归类于反向传播网络之中。

9.1.2 单层感知器功能

为便于直观分析,考虑图 9-2 中单计算节点感知器的情况。不难看出,单计算节点感知器实际上就是一个 M-P 神经元模型,由于采用了符号变换函数,又称为符号单元,它可表达为:

$$o_j = \begin{cases} 1 & W_j^T X > 0 \\ -1 & W_j^T X < 0 \end{cases}$$

下面分 3 种情况讨论单计算节点感知器的功能。

(1) 设输入向量 $X = (x_1, x_2)^T$, 则两个输入分量在几何上构成一个二维平面,输入样本可以用该平面上的一个点表示。节点 j 的输出为:

$$o_j = \begin{cases} 1 & w_{1j}x_1 + w_{2j}x_2 - T_j > 0 \\ -1 & w_{1j}x_1 + w_{2j}x_2 - T_j < 0 \end{cases}$$

则由方程

$$w_{1j}x_1 + w_{2j}x_2 - T_j = 0 \quad (9-1)$$

确定的直线成为二维输入样本空间的一条分界线。线上方的样本用*表示,它们使 $\text{net}_j > 0$, 从而使输出为 1; 线下方的样本用“○”表示,它们使 $\text{net}_j < 0$, 从而使输出为-1,如图 9-3 所示。显然,由感知器权值和阈值确定的直线方程规定了分界线在样本空间的位置,从而也确定了如何将输入样本分为两类。假如分界线的初始位置不能将“*”类样本同“○”类样本正确分开,改变权值和阈值,分界线也会随之改变,因此总可以将其调整到正确分类的位置。

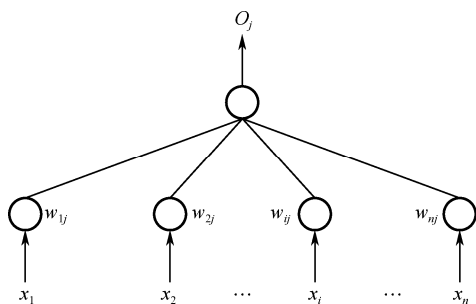


图 9-2 单计算节点感知器

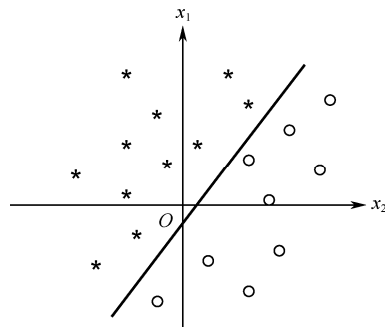


图 9-3 单计算节点感知器对二维样本的分类

(2) 设输入向量 $X = (x_1, x_2, x_3)^T$, 则 3 个输入分量在几何上构成一个三维空间。节点 j 的输出为:

$$o_j = \begin{cases} 1 & w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 - T_j > 0 \\ -1 & w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 - T_j < 0 \end{cases}$$

则由方程

$$w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 - T_j = 0 \quad (9-2)$$

确定的平面成为三维输入样本空间上的一个分界平面。平面上方的样本用“*”表示,它们使 $\text{net}_j > 0$, 从而使输出为 1; 平面下方的样本用“○”表示,它们使 $\text{net}_j < 0$, 从而使输出为-1。



同样，由感知器权值和阈值确定的平面方程规定了分界平面在样本空间的方向与位置，从而也确定了如何将输入样本分为两类。假如分界平面的初始位置不能将“*”类样本同“○”类样本正确分开，改变权值和阈值即改变了分界平面的方向与位置，因此总可以将其调整到正确分类的位置。

(3) 将上述两个特例推广到 n 维空间的一般情况，设输入向量 $X = (x_1, x_2, x_3)^T$ ，则 n 个输入分量在几何上构成一个 n 维空间，由方程

$$w_{1j}x_1 + w_{2j}x_2 + \cdots + w_{nj}x_n - T_j = 0 \tag{9-3}$$

可定义一个 n 维空间上的超平面，此平面可以将输入样本分为两类。

通过以上分析可以看出，一个最简单的单计算节点感知具有分类功能。其分类原理是将分类知识存储于感知器的权向量（包含了阈值）中，由权向量确定的分类判定界面将输入模式分为两类。

下面研究用单计算节点感知器实现逻辑运算问题。

首先，用感知器实现逻辑“与”功能。逻辑“与”的真值表及感知器结构如下：

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

从真值表中可以看出，4 个样本的输出有两种情况，一种使输出为 0，另一种使输出为 1，因此属于分类问题。用感知器学习规则进行训练，得到的连接权值标在图 9-4 中。令净输入为零，可得到分类判决方程为：

$$0.5x_1 + 0.5x_2 - 0.75 = 0$$

由图 9-5 可以看出，该方程确定的直线将输出为 1 的样本点“*”和输出为 0 的样本点“○”正确分开了。从图中还可以看出，该直线并不是唯一解。

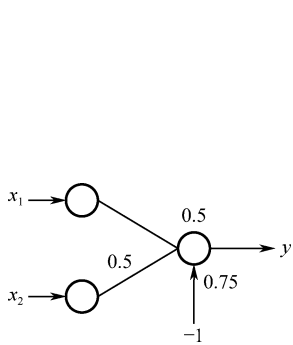


图 9-4 “与”逻辑感知器

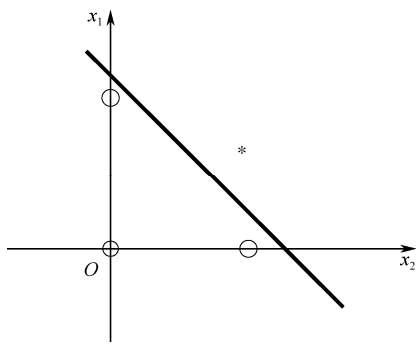


图 9-5 “与”运算分类

同样，可以用感知器实现逻辑“或”功能。逻辑“或”的真值表如下：

x_1	x_2	y
0	0	0
0	1	1

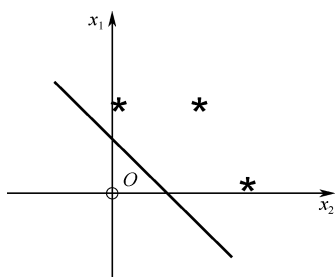


图 9-6 “或”运算的分类

1	0	1
1	1	1

从真值表中可以看出, 4 个样本的输出也分两类, 一类使输出为 0, 另一类使输出为 1。用感知器学习规则进行训练, 得到的连接权值为 $w_1 = w_2 = 1$, $T = -0.5$, 令净输入为零, 得分类判决方程为:

$$x_1 + x_2 - 0.5 = 0$$

该直线能把图 9-6 中的两类样本分开, 显然, 该直线也不是唯一解。

9.1.3 单层感知器结构

感知器网络由单层的 s 个感知神经元, 通过一组权值 $\{w_{ij}\} (i=1,2,\dots,s; j=1,2,\dots,r)$ 与 r 个输入相连组成。对于具有输入向量 $P_{r \times q}$ 和目标向量 $T_{s \times q}$ 的感知器网络, 其简化结构如图 9-7 所示。

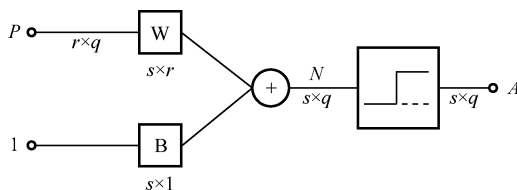


图 9-7 感知器网络简化结构图

根据网络结构, 可以写出第 i ($i=1,2,\dots,s$) 个输出神经元节点的加权输入和 n_i 及其输出 a_i 为:

$$n_i = \sum_{j=1}^r w_{ij} p_j + b_i \quad (9-4)$$

$$a_i = f(n_i) \quad (9-5)$$

感知器的输出值是通过测试加权输入和值落在阈值函数的左右来进行分类的, 即有:

$$a_i = \begin{cases} 1, & n_i \geq 0 \\ 0, & n_i < 0 \end{cases} \quad (9-6)$$

阈值激活函数如图 9-8 所示。

由图 9-8 可知, 当输入 $\sum_{j=1}^r w_{ij} p_j + b_i$ 大于等于 0, 即

$\sum_{j=1}^r w_{ij} p_j \geq -b_i$ 时, 感知器的输出为 1, 否则输出 a_i 为 0。利用偏差 b_i 使其函数可以左右移动, 从而增加了一个自由调整变量和实现网络特性的可能性。

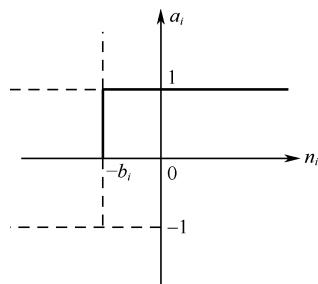


图 9-8 阈值激活函数



9.1.4 单层感知器学习算法

单层感知器对权值向量的学习算法是基于迭代的思想,通常采用纠错学习规则的学习算法。

为方便起见,将偏差 b 作为神经元突触权值向量的第一个分量加到权值向量中去,那么对应的输入向量也应增加一项,可设输入向量的第一个分量固定为+1,这样输入向量和权值向量可分别写成如下的形式:

$$X(n) = [+1, x_1(n), x_2(n), \dots, x_m(n)]^T \quad (9-7)$$

$$W(n) = [b(n), w_1(n), w_2(n), \dots, w_m(n)]^T \quad (9-8)$$

其中,变量 n 表示迭代次数, $b(n)$ 可用 $w_0(n)$ 表示,则二值阈值元件的输入可重新写为:

$$v = \sum_{j=0}^m w_j(n) x_j(n) = W^T(n) X(n) \quad (9-9)$$

令式(9-9)等于零,即 $W^T X = 0$ 可得在 m 维信号空间单层感知器的判决超平面。

学习算法如下:

(1) 设置变量和参量:

$X(n) = [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$ 为输入向量,或称训练样本。

$W(n) = [b(n), w_1(n), w_2(n), \dots, w_m(n)]^T$ 为权值向量。

$b(n)$ 为偏差。

$y(n)$ 为实际输出。

$d(n)$ 为期望输出。

η 为学习速率。

n 为迭代次数。

(2) 初始化,赋给 $w_j(0)$ 一个较小的随机非零值, $n = 0$ 。

(3) 对于一组输入样本 $X(n) = [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$, 指定它的期望输出 d (亦称之为导师信号)。如果 $X \in l_1$, $d = 1$; $X \in l_2$, $d = -1$ 。

(4) 计算实际输出:

$$y(n) = \text{sgn}(W^T(n) X(n)) \quad (9-10)$$

(5) 调整感知器的权值向量:

$$W(n+1) = W(n) + \eta[d(n) - y(n)]X(n) \quad (9-11)$$

(6) 判断是否满足条件:若满足,算法结束;若不满足,将 n 值增加 1,转到第(3)步重新执行。

注意:

在以上学习算法的第(6)步需要判断是否满足条件,这里的条件可以是:误差小于设定的值 ε , 即 $|d(n) - y(n)| < \varepsilon$; 或者是权值的变化已很小, 即 $|w(n+1) - w(n)| < \varepsilon$ 。另外,在实现过程中还应设定最大的迭代次数,以防止算法不收敛时,程序进入死循环。

在感知器学习算法中,重要的是引入了一个量化的期望输出 $d(n)$, 其定义为:



$$d(n) = \begin{cases} +1 & \text{如果 } X(n) \text{ 属于 } I_1 \\ -1 & \text{如果 } X(n) \text{ 属于 } I_2 \end{cases} \quad (9-12)$$

这样就可以采用纠错学习规则对权值向量进行逐步修正。

对于线性可分的两类模式，可以证明单层感知器的学习算法是收敛的，即通过学习调整突触权值可以得到合适的判决边界，正确区分两类模式，如图 9-9 (a) 所示。而对于线性不可分的两类模式，如图 9-9 (b) 所示，无法用一条直线区分两类模式。因而单层感知器的学习算法是不收敛的，即单层感知器无法正确区分线性不可分的两类模式。

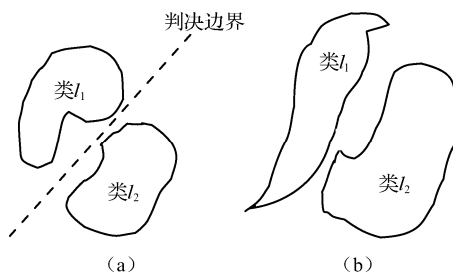


图 9-9 线性可分与不可分的问题

9.1.5 单层感知器训练

要使前身神经网络模型实现某种功能，必须对它进行训练，让它逐步学会要做的事情，并把所学到的知识记忆在网络的权值中。人工神经网络权值的确定不是通过计算，而是通过网络的自身训练完成的。这也是人工神经网络在解决问题的方式上与其他方法的最大不同点。借助于计算机的帮助，几百次甚至上千次网络权值的训练与调整过程能够在很短的时间内完成。

感知器的训练过程如下：

在输入向量 \mathbf{P} 的作用下，计算网络的实际输出 \mathbf{A} ，并与相应的目标向量 \mathbf{T} 进行比较，检查 \mathbf{A} 是否等于 \mathbf{T} ，然后用比较后的误差 \mathbf{E} ，根据学习规则进行权值和偏差的调整；重新计算网络在新权值作用下的输入，重复权值调整过程，直到网络的输出 \mathbf{A} 等于目标向量 \mathbf{T} 或训练次数达到事先设置的最大值时训练结束。

如果网络训练成功，那么训练后的网络在网络权值的作用下，对于被训练的每一组输入向量都能够产生一组对应的期望输出；如果在设置的最大训练次数内网络未能够完成在给定的输入向量 \mathbf{P} 的作用下，使 $\mathbf{A}=\mathbf{T}$ 的目标，则可以修改初始权值与偏差，并采用更长训练次数进行训练，或分析一下所要解决的问题是否属于那种由于感知器本身的限制而无法解决的一类。

感知器设计训练的步骤可总结为：

(1) 对于所要解决的问题，确定输入向量 \mathbf{P} ，目标向量 \mathbf{T} ，并由此确定各向量的维数以及确定网络结构大小的神经元节点数目 r 、 s 和 q 。

(2) 参数初始化：

- 赋给权向量 \mathbf{W} 在 $(-1,1)$ 的随机非零初始值。



● 给出最大训练循环次数。

(3) 网络表达式：根据输入向量 P 及最新权向量 W ，计算网络输出向量 A 。

(4) 检查：检查输出向量 A 与目标向量 T 是否相同，如果是，或已达到最大循环次数，训练结束，否则转入 (2)。

(5) 学习：根据感知器的学习规则调整权向量，并返回 (3)。

人工神经网络设计者的主要任务是确定三点内容：一是网络的类型，即确定选用什么样的典型神经元网络，是感知器还是后面所要学习的自适应线性元件，单层还是多层网络，目的是从众多网络中确定一种或几种网络的组合；二是网络的结构，包括网络输入和输出的节点数；三是网络的参数，即网络的权值。第三个任务是根据适当的算法训练出来的，而前两个任务是根据具体所要解决的问题转化来的，因为网络唯一的任务是处理数据，也只能处理数据，所以任何需要人工神经网络解决的问题都必须变成或者说是“预处理”成网络能够接受或称为认识的数值，再让网络来确定。

9.1.6 单层感知器局限性

由于感知器神经网络在结构和学习规则上的局限性，其应用被限制在一定的范围内。一般来说，感知器有以下局限性：

(1) 由于感知器的激活函数是强限幅传递函数，则感知器网络的输出值只能取 0 或 1。

(2) 感知器神经网络只能对线性可分的向量集合进行分类，如果可以用一条直线或者一个平面将输入向量分开，则称输入向量为线性可分的。如果输入向量不是线性可分的，网络学习就无法达到向量分界点。理论上已经证明，只要输入向量是线性可分的，感知器就能够在有限的循环内训练达到期望值。

(3) 当感知器神经网络的所有输入样本中存在奇异样本，即该样本向量同其他样本向量比较起来特别大或特别小时，网络训练所花费的时间将很长，例如输入样本：

```
p=[-0.5 -0.5 0.3 -0.1 -80; -0.5 0.5 -0.5 1.0 100];
t=[1 1 0 0 1]
```

其中，第五组数远远大于其他输入数据，这必然导致训练的困难，解决此问题的方法是采用标准化感知器学习规则。

1) 标准化感知器学习规则

如果输入向量存在奇异性，即奇异向量远远大于或者远远小于其他输入向量，则会导致网络训练的时间大大加长。因为当增加或者减少输入向量时，为了达到训练精度，都会改变网络的权值和阈值。而如果增加一个很大的样本，则会导致权值和阈值的很大改变，这就需要很长时间。为了解决此问题，提出了一种改进的感知器学习规则——标准化感知器学习规则。

原始的感知器学习规则采用式 (9-13) 进行权值调整：

$$\Delta w = (t - a)p^T = ep^T \quad (9-13)$$

从式 (9-13) 可见，输入向量 p 越大，权值的变化就越大。当存在奇异样本时，相对很



小的样本需要花很长的时间才能同奇异样本所对应的权值变化相匹配。因此，标准化感知器学习规则试图使奇异样本和其他样本对权值的变化值的影响均衡，即

$$\Delta w = (t - a) \frac{p^T}{\|p\|} = e \frac{p^T}{\|p\|}$$

在神经网络工具箱中，标准化感知器学习规则是由函数 `learpn` 实现的。

标准化感知器学习规则相对于原始感知器学习规则来说，网络的训练时间稍长，但是对于含有奇异样本的输入向量，标准化感知器学习规则又是非常有效的。

2) 多层感知器

为了解决单层感知器网络的局限性，20 世纪 60 年代末期人们致力于该问题的研究，并找到了解决的方法——多层感知器神经网络，即网络中包含一个以上的感知器神经元，这样一些困难的问题就迎刃而解了。例如，假设要将四个向量分成两类，可以画两条线分开它们。如果在网络中设置两个神经元，就有了两条分界线，完全可以将四个输入向量分成两类，这就是多层感知器神经网络。

图 9-10 就是一个常用的双层感知器神经网络，第一层是随机感知层，第二层为学习感知层。

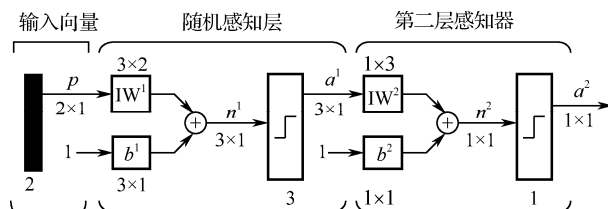


图 9-10 双层感知器网络结构图

9.1.7 单层感知器的 MATLAB 实现

下面通过具体示例来演示单层感知器在 MATLAB 中的应用。

【例 9-1】 给定样本输入向量 P ，目标向量 T 及需要进行分类的输入向量组 Q ，设计一个单层感知器，对其进行分类。

```
>>clear all;
P=[-0.6 -0.7 0.8;0.9 0 1];
T=[1 1 0];
net=newp([-1 1; -1 1],1);
%返回画线的句柄，下一次绘制分类线时将旧的删除
he=plotpc(net.iw{1},net.b{1});
%设置训练次数最大为 15
net.trainParam.epochs=15;
net=train(net,P,T);
%给定的输入向量
```

由图 9-11 可见，所设计的感知器对输入模式进行了成功的分类。
经过 15 次训练后，网络目标误差达到要求，如图 9-12 所示。

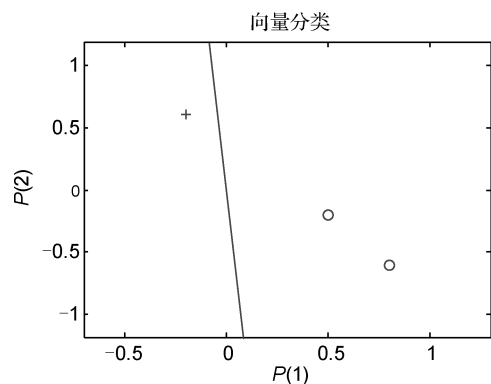


图 9-11 输入向量及分类线

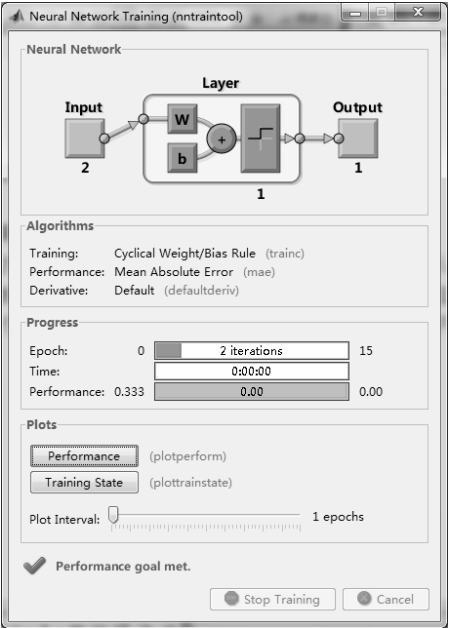


图 9-12 网络训练过程

```
Q=[0.5 0.8 -0.2; -0.2 -0.6 0.6];
Y=sim(net,Q);
figure;
%绘制分类线
plotpv(Q,Y);
he=plotpc(net.iw{1},net.b{1},he)
title('向量分类');
```

运行程序，输出如下，效果如图 9-11 所示。

```
he =
    174.0071
```

【例 9-2】 某工厂在厂区附近对大气质量进行检测，取 8 份样品进行分析，检测结果如表 9-1 所列。现有 2 份取自该区的气体样品，请判别这 2 份气体样品的污染分类。

表 9-1 大气样品数据表

气 体	氯	硫 化 氢	二 氧 化 硫	碳 4 化 合 物	环 己 烷	污 染 分 类
1	0.056	0.084	0.031	0.038	0.022	1
2	0.040	0.055	0.100	0.110	0.073	1
3	0.030	0.090	0.068	0.180	0.039	1
4	0.069	0.084	0.027	0.050	0.089	1

续表

气 体	氯	硫 化 氢	二 氧 化 硫	碳 4 化合物	环 己 烷	污 染 分 类
5	0.084	0.066	0.029	0.320	0.041	2
6	0.064	0.072	0.020	0.250	0.038	2
7	0.038	0.130	0.079	0.170	0.043	2
8	0.048	0.089	0.062	0.260	0.036	2
样品 1	0.052	0.084	0.021	0.037	0.022	
样品 2	0.074	0.083	0.105	0.190	1.000	

其实现的 MATLAB 代码为:

```
>> clear all;
p=[0.056 0.040 0.030 0.069 0.084 0.064 0.038 0.048;0.084 0.055 0.090 0.084 0.066 0.072...
    0.130 0.089;0.031 0.100 0.068 0.027 0.029 0.020 0.079 0.062;0.038 0.110 0.180 0.050...
    0.320 0.250 0.170 0.260;0.022 0.073 0.039 0.089 0.041 0.038 0.043 0.036];
t=[0 0 0 1 1 1 1]; %污染等级 1 设为 0, 等级 2 设为 1
net=newp([0 0.5;0 0.5;0 0.5;0 0.5;0 0.5],1); %创建 5 个输入单神经元的感知器
net=init(net);
net.trainParam.epochs=500; %设置训练步数
[net,tr]=train(net,p,t); %训练, 结果显示训练 151 步, 达到误差要求
p1=[0.0520 0.0740;0.0840 0.0830;0.0210 0.1050;0.0370 0.1900;0.022 1.000]; %未知样品
t1=sim(net,p1) %对未知样品仿真
```

运行程序, 输出如下, 效果如图 9-13 所示。

```
t1 =
    0    0
```

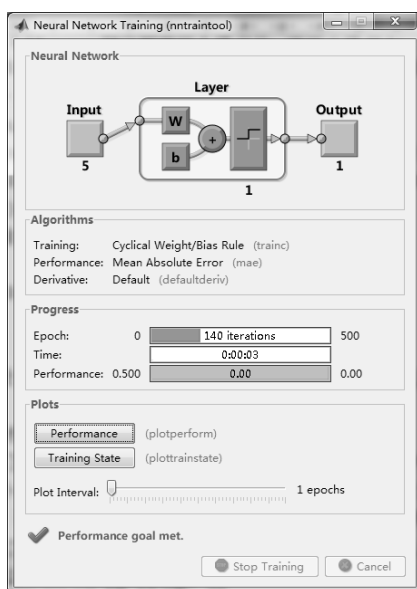


图 9-13 网络训练过程



结果表明，两个未知气体样品的污染等级都为一类，完全符合实际。

【例 9-3】设计一个单层双输出感知器神经网络，进行二值化图像卡片上数字的奇偶分类。

1) 问题分析和神经网络设计

图 9-14 给出了数字 1 和 0 的二值化图像卡片，每一个图像卡片可以分成 5×3 的矩形方块，假设每个小方块有数字的笔画经过（即在小方块内二值图像元素的值至少有一个不为 0），则记为 1，否则记为 0，那么图像卡片上所有小方块表达了有 0, 1 二值组成的一个模式（或向量），该模式可以作为感知器神经网络的输入向量。

本例要求用两个输出端表示分类结果，设 $a_1 a_2 = 10$ ，表示偶数； $a_1 a_2 = 01$ 表示奇数；则 $a_1 a_2 = 00$ 为拒识状态； $a_1 a_2 = 11$ 为错误状态。设计的感知器神经网络结构示意图如图 9-15 所示。



图 9-14 图像数字卡片构成模式（向量）示意图

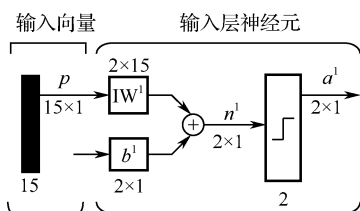


图 9-15 单层感知器神经网络模型

2) 感知器神经网络的 MATLAB 实现代码

```
>> clear all;
%设置输入向量每个元素的值域（最小值和最大值），因为有 15 个输入,所以为 15x2 矩阵向量
pr=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
%训练感知器神经网络
%定义 15x10 的训练样本集输入向量
p=[1 1 1 1 0 1 1 0 1 1 0 1 1 1 1;1 1 0 0 1 0 0 1 1 1 0 1 0 1 1;...
    1 1 1 1 0 1 0 1 1 1 1 0 1 1 1;1 1 1 1 1 1 0 1 1 0 0 1 1 1 1;...
    0 1 0 1 1 0 1 1 0 1 1 1 0 1 0;1 1 1 1 1 0 0 1 1 0 0 1 1 1 1;...
    0 1 1 1 1 0 1 1 1 1 0 1 1 1 1;1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0;...
    1 1 1 1 0 1 1 1 1 1 0 1 1 1 1;1 1 1 1 0 1 1 1 1 0 1 1 1 1 0];

net=newp(pr,2); %创建感知器神经网络,有 15 个输入元素, 2 个神经元
t=[1 0 1 0 1 0 1 0 1 0;0 1 0 1 0 1 0 1 0 1]; %定义 2x10 的目标向量
[net,tr]=train(net,p,t); %训练单层感知器神经网络
iw1=net.IW{1} %输出训练后的权值
b1=net.b{1} %输出训练后的阈值
epoch1=tr.epoch %输出训练过程经过的每一步长
perf1=tr.perf %输出每一步训练结果的误差
%存储训练后的神经网络
save netli48 net
```

运行程序，输出如下：

```
iw1 =
Columns 1 through 12
    -2     0    -1     1    -2    -1     3    -1    -1     6     1     0
     2     0     0    -1     1     1    -3     2     0    -5     1    -1

Columns 13 through 15
    -1     0     0
     0     0    -1

b1 =
     0
     0

epoch1 =
     0     1     2     3

perf1 =
    0.5000    0.3000    0.3000     0
```

训练误差效果图如图 9-16 所示。

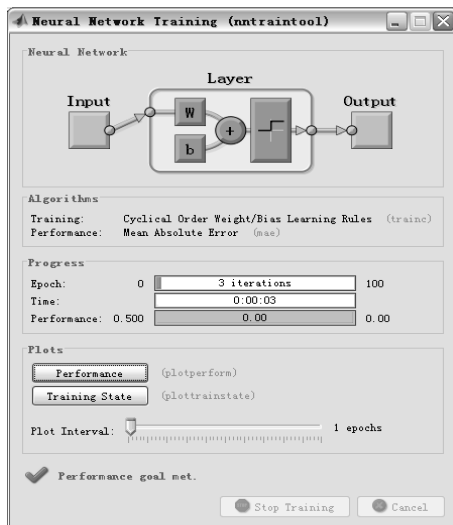


图 9-16 训练误差效果图

从运行结果可以看出，本例所设计的单层双输出神经网络经过 3 步训练后就达到误差为 0 的性能指标。

实现感知器神经网络仿真的 MATLAB 程序代码如下：

```
>> clear all; %清除所有内存变量
load netli48 net %加载训练后的神经网络
%对训练后的神经网络进行仿真
ptest=[1 1 1 0 1 1 0 1 1 0 1 1 0 1;... %数字 0, 与训练样本不一致
        1 1 0 0 1 0 0 1 1 0 1 0 1 1 1;... %数字 1, 与训练样本一致
        1 1 1 0 1 1 0 1 0 1 1 1 0 0 0;... %非数字
        1 1 1 1 1 0 1 0 1 0 0 0 1 1 1]; %非数字
```




```
a=sim(net,ptest)
```

```
%输出仿真结果
```

进行仿真的结果如下：

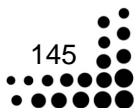
```
a =
     1     0     1     0
     0     1     1     0
```

可以看出，对于训练样本及训练样本以外的输入模式可以得到正确的分类结果，具有一定的容错能力。因为有两个输出端，还可以表示拒识（识别结果偶数和奇数都不是）和错误（识别结果同为奇数和偶数）的情况。

【例 9-4】 给出感知器面对线性不可分输入/输出模式时的情况。

由于感知器对输入向量空间只能线性地进行输出 0 或 1 的分类，故它们只能适当地划分具有线性可分性的输入向量组。如果输入向量组和它所对应的 0、1 目标之间不能用直线进行划分，那么感知器就不能正确地分类输入向量。

```
>> clear all;
%线性不可分输入向量
P=[-0.5 -0.5 0.3 0 -0.8; -0.5 0.5 -0.5 1 0];
%目标向量
T=[1.0 1.0 0 0 0];
V=[-2 2 -2 2];
net=newp(minmax(P),1,'hardlim','learnp'); %创建一个感知器网络
net.inputweights{1,1}.initFcn='rands'; %赋输入权值的产生函数
net.biases{1}.initFcn='rands'; %赋偏差的产生函数
net=init(net); %网络初始化
disp('输出初始化权值: ')
W0=net.iw{1,1}
disp('输出初始化偏差: ')
B0=net.b{1}
A=sim(net,P); %网络仿真
net.trainParam.epochs=40;
[net,tr]=train(net,P,T); %训练网络权值
W=net.iw{1,1};
B=net.b{1,1};
plotpv(P,T,V);
hold on;
plotpc(W0,B0); %绘制分类线曲线
plotpc(W,B);
hold off;
fprintf('\n 最终的网路值: \n')
W
B
fprintf('最大训练次数: ')
```



```
max(tr.epoch)
fprintf('\n 网络分类: ')
if all(hardlim(W*P+B)==T)
    disp('可分模式');
else
    disp('不可分模式');
end
```

运行程序，输出如下，效果如图 9-17 及图 9-18 所示。

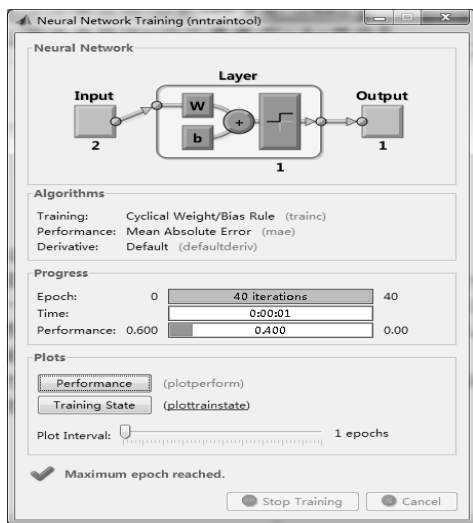


图 9-17 感知器网络训练过程

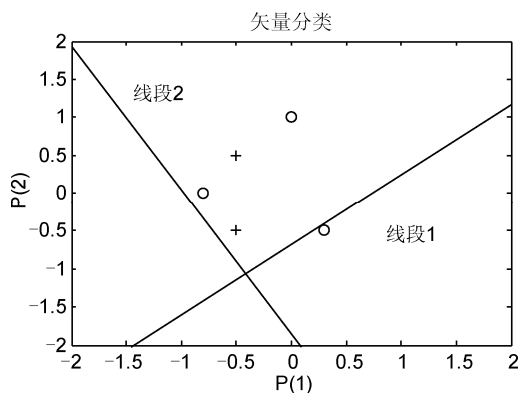


图 9-18 线性不可分模式训练

输出初始化权值:

```
W0 =
    0.7899   -0.8571
```

输出初始化偏差:

```
B0 =
   -0.5748
```

最终的网路值:

```
W =
   -1.6101   -0.8571
B =
   -1.5748
```

最大训练次数:

```
ans =
    40
```



网络分类：不可分模式。

在感知器应用中，线性不可分问题都归结为“异或”问题。

9.2 多层感知器

9.2.1 多层感知器模型

多层感知器是对单层感知器的推广，它能够成功解决单层感知器所不能解决的非线性可分问题，其拓扑结构如图 9-19 所示。

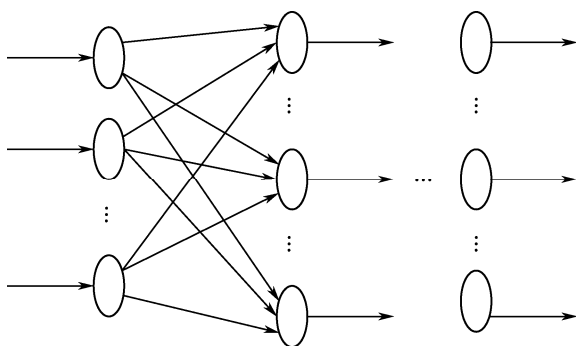


图 9-19 多层感知器拓扑结构

(1) 多层感知器含有一层或多层隐单元。隐单元从输入模式中获得了更多有用的信息，使网络可以完成更复杂的任务。

(2) 多层感知器中每个神经元的激活函数采用可微的函数，如 sigmoid 函数

$$v_i = \frac{1}{1 + \exp(-u_i)}$$

式中， u_i ——第 i 个神经元的输入信号；

v_i ——该神经元的输出信号。

(3) 多层感知器的多个突触使得网络更具连通性，连接域的变化及连接权值的变化都会引起连通性的变化。

(4) 多层感知器具有独特的学习算法，该学习算法就是著名的 BP 算法，因此使用 BP 算法的神经网络也常常被称为 BP 网络。

多层感知器所具有的这些特点，使得它具有强大的计算能力，从而成为一种广泛使用的神经网络。

9.2.2 多层感知器设计方法

单层感知器由于其结构和学习规则上的局限性，其应用也受到一定的限制，即它只能对



线性可分的向量集合进行分类。为了解决线性不可分的输入向量分类问题，可以增加网络层。

由于感知器神经网络学习规则的限制，它只能对单层感知器神经网络进行训练，那么，如何进行多层感知器的神经网络设计呢？这里提供了一种二层神经感知器神经网络的设计方法：

(1) 把神经网络的第一层设计为随机感知层，且不对它进行训练，而是随机初始化它的权值和阈值，当它接收各输入元素的值时，其输出也是随机的，但其权值和阈值一旦固定下来，对输入向量模式的映射也随之确定下来。

(2) 以第一层的输出作为第二感知器层的输入，并对应输入模式，确定第二感知层的目标向量，然后对第二感知器层进行训练。

(3) 由于第一随机感知器层的输出是随机的，所以在训练过程中，整个网络可能达到训练误差性能指标，也可能达不到训练误差性能指标。所以，当达不到训练误差指标，需要重新对随机感知器层的权值和阈值进行初始化赋值，可以将其初始化函数设置为随机函数，然后用 `init` 函数重新初始化。程序一次运行的结果往往达不到设计要求，需要反复运行，直至达到要求为止。

9.2.3 多层感知器的 MATLAB 实现

由于神经网络工具箱中没有为多层感知器提供专门的函数，似乎没有办法利用 MATLAB 实现多层感知器了，其实不然，MATLAB 的神经网络工具箱中的定制网络功能可以帮助我们解决这个问题。

【例 9-5】 创建一个未知的新网络。

```
>> clear all;
net=network;
%y 设置网络的结构属性
net.numInputs=1;
net.numLayers=2;           %输入层的数目
%输入层之间的连接情况
net.biasConnect=[1;1];     %设置网络的阈值
net.inputConnect=[1;0];    %输入权值
net.layerConnect=[0 0;1 0]; %层权值
net.outputConnect=[0 1];   %输出连接情况
%设置各层的属性
net.inputs{1}.range=[0 1;0 1]; %输出向量范围
net.layers{1}.size=2;        %输入层数大小
net.layers{1}.transferFcn='hardlim'; %传递函数
net.layers{1}.initFcn='initnw'; %初始化函数
net.layers{2}.size=1;        %输入层数大小
net.layers{2}.transferFcn='hardlim'; %传递函数
net.layers{2}.initFcn='initnw'; %初始化函数
%设置网络的整体属性
```



```

net.adaptFcn='trainlm';           %自适应函数
net.performFcn='mse';            %性能函数
net.trainFcn='trainlm';         %训练函数
net.initFcn='initlay';          %初始化函数
%设置训练参数对其进行训练，利用其实现异或功能
%对网络进行初始化
net=init(net);
%设置训练样本 P 和 T
P=[0 0;0 1;1 0;1 1];
T=[0 1 1 0];
%设置网络训练次数 500
net.trainParam.epochs=500;
net=train(net,P,T);

```

得到训练过程如图 9-20 所示。

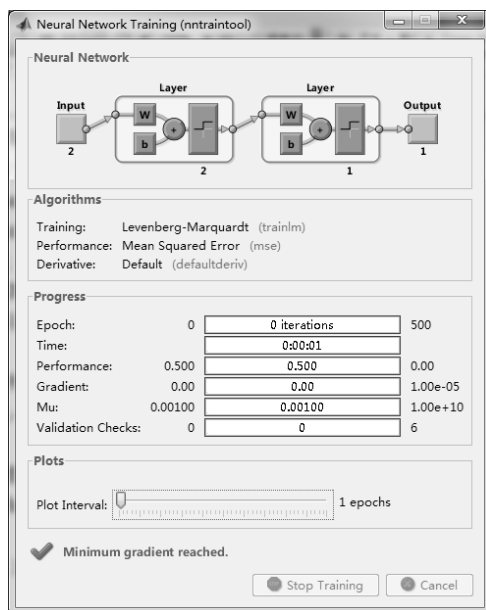


图 9-20 多层感知器网络训练过程

检验网络的性能是否达到要求，代码为：

```
>> y=sim(net,P)
```

运行程序，输出如下：

```

y =
    0     1     1     1

```

可见，此时的网络已经实现了异或功能。

【例 9-6】 单层感知器网络不能模拟异或函数的问题，这里用二层感知器神经网络来实现。

(1) 问题分析。

异或问题真值表见表 9-2。

表 9-2 异或问题真值表

输入 p_1	输入 p_2	输出 a
0	0	0
1	0	1
0	1	1
1	1	0

若把异或问题看成 $p_1 - p_2$ 平面上的点，则点 $A_0(0,0)$ ， $A_1(1,1)$ 表示输出为 0 的两个点， $B_0(0,0)$ ， $B_1(1,1)$ 表示输出为 1 的两个点，如图 9-21 所示。

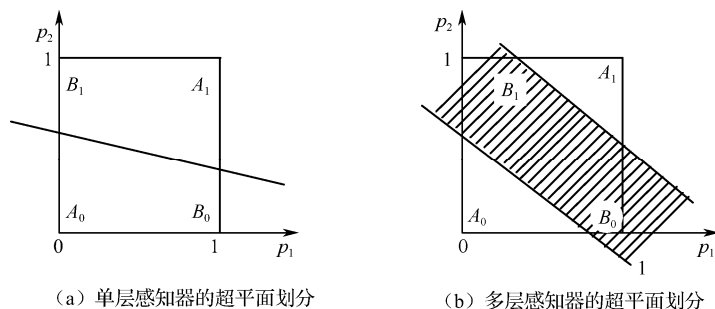


图 9-21 异或问题的图形表示

从图中可以看出，无论在平面上怎样用一条直线也不可能将输出为 0 和 1 的两种模式分开，而用两条直线就能将输出为 0 和 1 的两种模式分开。

(2) 神经网络的设计。

根据以上分析，如果用两层感知器，每层感知器可以构成一条直线划分，则可以解决模拟异或函数的问题。以图 9-10 所示双层神经网络来实现，其中隐层为随机感知器层 (net1)，神经元数目设计为 3，其权值和阈值是随机的，它的输出作为输出层 (分类层) 的输入；输出层为感知器层 (net2)，其神经元数为 1，这里仅对该层进行训练。

(3) 多层感知器神经网络的 MATLAB 实现，其代码如下。

```
>>clear all;
%初始化随机感知器层
PR1=[0 1;0 1];
net1=newp(PR1,3);
net1.inputweights{1}.initFcn='rands';
net1.biases{1}.initFcn='rands';
net1=init(net1);
IW1=net1.iw{1}
B1=net1.b{1}

%随机感知器层仿真量
```



```

P1=[0 0;0 1;1 0;1 1];
[A1,Pf]=sim(net1,P1);

%初始化第二感知器层
PR2=[0 1;0 1;0 1];
net2=newp(PR2,1);

%训练第二感知器层
net2.trainParam.epochs=10;
net2.trainParam.show=1;
P2=ones(3,4);
P2=P2.*A1;
T2=[0 1 1 0];
[net2,TR2]=train(net2,P2,T2);
epoch2=TR2.epoch
perf2=TR2.perf
IW2=net2.iw{1}
B2=net2.b{1}
%存储训练后的网络
save net34 net1 net2

```

因为随机感知器层的输出是随机的，所以整个网络可能达到训练误差指标，也可能达不到训练误差指标。因此，当达不到训练误差指标时，需要重新对随机感知器层的权值和阈值进行初始化赋值，在本例程序中，是通过将其初始化函数设置为随机函数，然后用 `init` 函数重新初始化来实现的。该程序一次运行的结果可能达不到设计要求，需要反复运行，直至达到要求为止。正因为如此，每次训练得到的结果也不尽相同。

这样做是因为设计受到感知器学习算法的限制，对于多层网络，当然有更好的学习算法，比如后面将要介绍的 BP 网络学习算法。

其中达到训练误差指标的一种运行结果如下：

```

IW1 =
    0.7200   -0.0069
    0.7073    0.7995
    0.1871    0.6433
B1 =
   -0.6983
    0.3958
   -0.2433
epoch2 =
     0     1     2     3     4     5     6     7     8     9    10
perf2 =
    0.5000    0.7500    0.7500    0.5000    0.5000    0.7500    1.0000
    1.0000    0.2500    0.5000    0.5000
IW2 =

```

```
-3    2    -2
B2 =
     2
```

训练误差性能过程如图 9-22 所示。

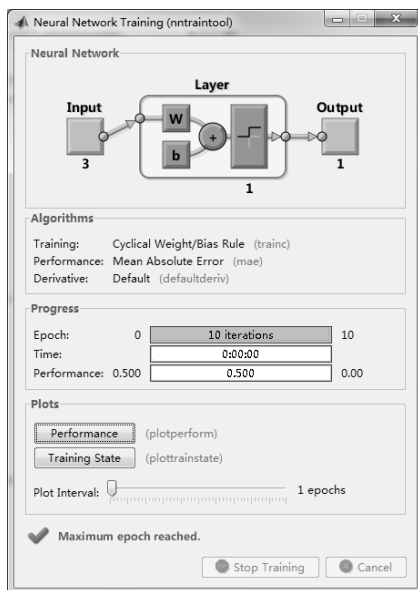


图 9-22 训练误差性能过程

下面为多层感知器神经网络仿真的 MATLAB 程序。

```
clear all;
%加载训练后的网络
load net34 net2

%随机感知器层仿真
P1=[0 0;0 1;1 0;1 1];
A1=sim(net1,P1);

%输出感知器层仿真，并输出仿真结果
P2=ones(3,4);
P2=P2.*A1;
A2=sim(net2,P2)
```

运行程序，输出如下：

```
A2 =
     1     1     1     0
```

可以看出，所设计的网络可以正确模拟“异或”函数的功能。

第 10 章 神经网络工具箱函数分析与应用



快速发展的 MATLAB 软件为神经网络理论的实现提供了一种便利的仿真手段。MATLAB 神经网络工具箱的出现，更增加了神经网络的应用空间。神经网络工具箱将很多原本需要手动计算的工作交给计算机，一方面提高了工作效率，另一方面，还提高了计算机的准确度和精度，减轻了工程人员的负担。

最新版本的神经网络工具箱几乎涵盖了所有神经网络的基本常用模型，如感知器和 BP 网络等。对于各种不同的网络模型，神经网络工具箱集成了多种学习算法，为用户提供了极大的方便。另外，该工具箱中还包括大量的演示示例，有助于初学者加深理解。

目前，神经网络工具箱中提供的神经网络模型类别主要有：

- 感知器网络；
- 线性网络；
- BP 网络；
- 动态网络；
- 径向基函数网络；
- 自组织映射和向量量子化网络。

目前，神经网络工具箱中提供的神经网络模型主要应用于：

- 函数逼近和模型拟合；
- 信息处理和预测；
- 神经网络控制；
- 故障诊断。

在实际应用中，面对一个具体的问题，首先需要分析利用神经网络求解问题的性质，然后依据问题特点，提取训练和测试数据样本，确定网络模型。最后通过对网络进行训练、仿真等，检验网络的性能是否满足要求。

10.1 神经网络仿真函数

在 MATLAB 神经网络工具箱中提供了 `sim` 函数，用于实现神经网络的仿真，其调用格式为：

```
[Y,Pf,Af,E,perf] = sim(net,P,Pi,Ai,T)
[Y,Pf,Af] = sim(net,{Q TS},Pi,Ai)
```

其中，

Y 为函数返回值，网络输出；



Pf 为函数返回值，最终输出延迟；

Af 为函数返回值，最终的层延迟；

E 为函数返回值，网络误差；

perf 为函数返回值，网络性能；

net 为待仿真的神经网络；

P 为网络输入；

Pi 为初始输入延迟，默认为 0；

Ai 为初始的层延迟，默认为 0；

T 为网络目标，默认为 0。

Pi、Ai、Pf 和 Af 是可选的，它们只用于存在输入延迟和层延迟的网络。

函数中用到的信号参数采取了两种不同的形式表示，分别为单元阵列和矩阵的形式。其中单元阵列能够方便地对多输入、多输出的神经网络进行描述。信号参数为单元阵列下的输入/输出形式如下所述。

P: $N_i \times TS$ 维的单元阵列，每个元素 $P\{i, ts\}$ 都是一个 $R_i \times Q$ 的矩阵；

Pi: $N_i \times ID$ 维的单元阵列，每个元素 $P\{i, k\}$ 都是一个 $R_i \times Q$ 的矩阵；

Ai: $N_i \times LD$ 维的单元阵列，每个元素 $Ai\{i, k\}$ 都是一个 $S_i \times Q$ 的矩阵；

T: $N_t \times TS$ 维的单元阵列，每个元素 $P\{i, ts\}$ 都是一个 $V_i \times Q$ 的矩阵；

Y: $N_o \times TS$ 维的单元阵列，每个元素 $Y\{i, ts\}$ 都是一个 $U_i \times Q$ 的矩阵；

Pf: $N_i \times ID$ 维的单元阵列，每个元素 $Pf\{i, k\}$ 都是一个 $R_i \times Q$ 的矩阵；

Af: $N_i \times LD$ 维的单元阵列，每个元素 $Af\{i, k\}$ 都是一个 $S_i \times Q$ 的矩阵；

E: $N_t \times TS$ 维的单元阵列，每个元素 $P\{i, ts\}$ 都是一个 $V_i \times Q$ 的矩阵；

其中，

N_i : 神经网络输入的数目；

N_l : 神经网络层次的数目；

N_o : 神经网络输出的数目；

ID: 输入延迟的数目；

LD: 层次延迟的数目；

TS: 时间步长的数目；

Q: 批量；

R_i : 第 i 个输入的长度；

S_i : 第 i 层的长度；

U_i : 第 i 个输出的长度；

$Pi\{i, k\}$: 第 i 个输入在 $ts=k-ID$ 时刻的状态；

$Pf\{i, k\}$: 第 i 个输入在 $ts=TS+k-ID$ 时刻的状态；

$Ai\{i, k\}$: 某层输出 i 在 $ts=k-LD$ 时刻的状态；

$Af\{i, k\}$: 某层输出 i 在 $ts=TS+k-LD$ 时刻的状态；

矩阵的形式只用于仿真时间步长 $TS=1$ 的场合，它适用于单输入/单输出的网络，但也同样适用于多输入/多输出的网络。在矩阵形式下，每个矩阵都是由对应的单元阵列中的元素组合而成的。其中，



P: R_i 的总和 $\times Q$ 维矩阵;
 Pi: R_i 的总和 $\times (ID \times Q)$ 维矩阵;
 Ai: S_i 的总和 $\times (LD \times Q)$ 维矩阵;
 T: V_i 的总和 $\times Q$ 维矩阵;
 Y: U_i 的总和 $\times Q$ 维矩阵;
 Pf: R_i 的总和 $\times (ID \times Q)$ 维矩阵;
 Af: S_i 的总和 $\times (LD \times Q)$ 维矩阵;
 E: V_i 的总和 $\times Q$ 维矩阵。

[Y,Pf,Af] = sim(net,{Q TS},Pi,Ai): 常用于一些没有输入信号的回归神经网络, 如 Hopfield 网络等。

【例 10-1】 对创建的神经网络进行仿真, 并绘制其对应效果图。

```
>> clear all;
p = [0 1 2 3 4 5 6 7 8]; %输入向量
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99]; %目标向量
%创建一个两层前馈网络。网络隐层有十个神经元
net = feedforwardnet(10);
net = configure(net,p,t);
y1 = sim(net,p) %对所创建网络进行仿真
plot(p,t,'o',p,y1,'x')
legend('训练前数据','训练后数据');
```

运行程序, 输出如下, 效果如图 10-1 所示。

```
y1 =
    1.7040    0.5532   -0.6375   -0.5379   -1.1686   -2.8748   -2.6469   -1.8394   -2.4301
```

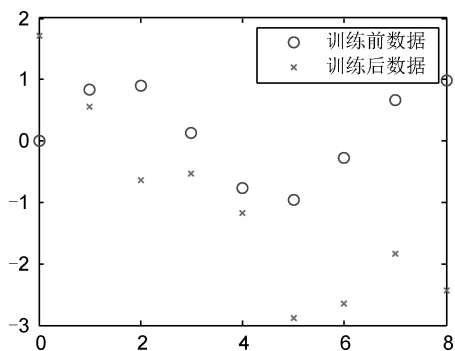


图 10-1 网络仿真效果图

10.2 神经网络训练函数

在 MATLAB 神经网络工具箱中提供了 train 及 trainb 函数, 用于实现神经网络的训练, 下面分别对这两个函数进行介绍。



10.2.1 train

该函数用于对神经网络进行训练，其调用格式为：

```
[net,tr,Y,E,Pf,Af] = train(net,P,T,Pi,Ai)
```

其中，

net: 输入参数，待训练的神经网络；

P: 网络的输入信号；

T: 网络的目标，默认为 0；

Pi: 初始的输入延迟，默认为 0；

Ai: 初始的层次延迟，默认为 0；

输出参数说明如下。

net: 函数返回值，训练后的神经网络；

tr: 函数返回值，训练记录（包括步数和性能）；

Y: 函数返回值，神经网络输出信号；

E: 函数返回值，神经网络的误差；

Pf: 函数返回值，最终输入延迟；

Af: 函数返回值，最终层延迟。

【例 10-2】 在例 10-1 的基础上对数据进行网络训练。

```
net.trainParam.epochs = 50;  
net.trainParam.goal = 0.01;  
net = train(net,p,t);  
y2 = sim(net,p)  
plot(p,t,'o',p,y1,'x',p,y2,'*')  
legend('原始数据','仿真后数据','训练后数据');
```

运行程序，输出如下，效果如图 10-2 及图 10-3 所示。

```
y2 =  
-0.0244  0.4645  0.9094  0.1398 -0.7694 -0.9625 -0.2778  0.6616  0.1640
```

10.2.2 trainb 函数

该函数用于神经网络权值和阈值的训练，其调用格式为：

```
[net,TR] = trainb(net,TR,trainV,valV,testV)  
info = trainb('info')
```

其中，

net: 待训练的神经网络；

Pd: 已延迟的输入信号；

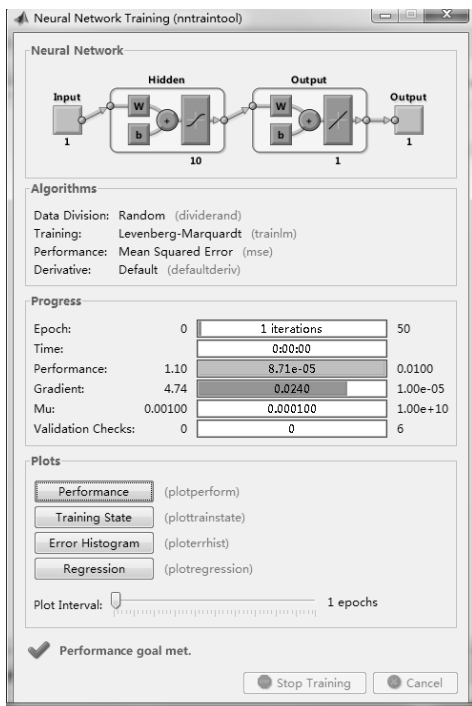


图 10-2 网络的训练过程

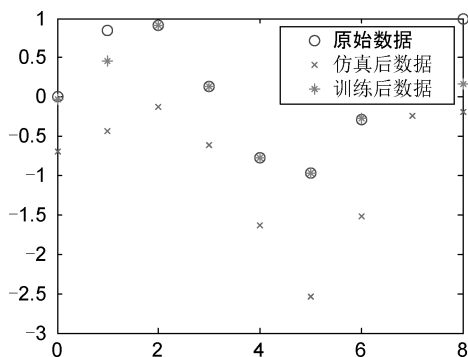


图 10-3 网络训练效果图

T1: 层目标;

Ai: 初始的输入;

Q: 批量;

TS: 时间步长;

VV: 确认向量或者为空矩阵;

TV: 测试向量或者为空矩阵;

NET: 函数返回值, 训练后的神经网络;

TR: 函数返回值, 每一步的训练记录;

Ac: 训练停止后, 聚合层的输出;

E1: 训练停止后的误差。

函数的调用形式 `info=trainb(code)` 将返回训练函数的有关信息, `code` 字符串的取值为:

- 当 `code=names` 时返回训练参数的名称;
- 当 `code=pdefaults` 时返回训练参数的默认值。

网络属性设置:

`newlin` 函数建立的网络采用 `trainb` 作为训练函数, 如果要使用自定义网络利用 `trainb` 函数进行训练, 可设置如下:

- (1) 将网络训练函数 `net.trainFcn` 设置为 'trainb';
- (2) 设置网络训练函数的参数 `net.trainParam`;
- (3) 设置各输入权值、网络权值和阈值的学习函数 `net.inputWeights{i,j}.learnFcn`、



`net.layerWeights{i,j}.learnFcn` 和 `net.biases{i}.learnFcn`;

(4) 设置各权值和阈值学习函数的参数 `net.inputWeights{i,j}.learnParam`、`net.layerWeights{i,j}.learnParam` 和 `net.biases{i}.learnParam`。

另外，训练之前需要设定以下参数，默认值如表 10-1 所列。

表 10-1 训练参数

参 数 名 称	默 认 值	属 性
<code>net.trainParam.epochs</code>	100	训练次数。100 为训练次数的最大值，人工设定的训练次数不能超过 100
<code>net.trainParam.goal</code>	0	训练目标
<code>net.trainParam.time</code>	inf	训练时间，inf 表示训练时间不限
<code>net.trainParam.max_fail</code>	5	最大确认失败次数
<code>net.trainParam.show</code>	25	两次显示之间的训练步数（无显示时取 NaN）

10.3 神经网络学习函数

在 MATLAB 神经网络工具箱中提供了若干函数用于实现神经网络的学习，下面分别给予介绍。

1. learnp 函数

该函数用于神经网络权值和阈值的学习，其调用格式为：

```
[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnp('code')
```

其中，

W: $S \times R$ 维的权值矩阵（或 $S \times 1$ 维的阈值向量）；

P: Q 组 R 维的输入向量（或 Q 组单个输入）；

Z: Q 组 S 维的权值输入向量；

N: Q 组 S 维的网络输入向量；

A: Q 组 S 维的输出向量；

T: Q 组 S 维的目标向量；

E: Q 组 S 维的误差向量；

gW: $S \times R$ 维的性能参数的梯度；

gA: Q 组 S 维的性能参数的输出梯度；

LP: 学习参数，如果没有则为空；

LS: 学习状态，初始值为空；

dW: $S \times R$ 维权值（或阈值）的变化矩阵；

LS: 新的学习状态。



在 `info = learnp('code')` 中, 针对不同的 `code` 返回相应的有用信息, 包括:

- 当 `code= pnames` 时表示返回函数全称;
- 当 `code=pdefaults` 时表示返回默认的训练参数;
- 当 `code=needg` 时表示如果函数使用了 `gW` 或 `gA`, 则返回 1。

【例 10-3】 对给定的输入向量及误差向量, 求其学习矩阵。

```
>> p = rand(2,1);
e = rand(3,1);
dW = learnp([],p,[],[],[],[],e,[],[],[],[])
```

运行程序, 输出如下:

```
dW =
    0.0516    0.0876
    0.1873    0.3182
    0.0818    0.1390
```

2. learnpn 函数

该函数也是一个权值和阈值学习函数, 但它在输入向量的幅值变化非常大或者存在奇异值时, 学习速度比 `learnp` 要快得多, 其调用格式为:

```
[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnpn('code')
```

参数含义与 `learnp` 相同。

【例 10-4】 一个三输入两神经元网络层的输入向量 `P` 和误差向量 `E` 分别为:

```
>> P=[0.1597;0.6932;0.4521];
E=[1;-1];
dW=learnpn([],P,[],[],[],[],E,[],[],[],[])
```

运行程序, 输出如下:

```
dW =
    0.1221    0.5300    0.3457
   -0.1221   -0.5300   -0.3457
```

3. adapt 函数

该函数使得神经网络能够自适应, 其调用格式为:

```
[net,Y,E,Pf,Af,tr] = adapt(net,P,T,Pi,Ai)
```

其中,

`net`: 输入参数, 为未自适应的神经网络;

`P`: 网络输入;

`T`: 网络目标, 默认为 0;

`Pi`: 初始输入延迟, 默认为 0;



Ai: 初始层延迟, 默认为 0。

通过设定自适应的参数 `net.adaptParam` 和自适应的函数 `net.adaptFunc` 可调用该函数, 并返回如下参数。

`net`: 输出参数, 自适应后的神经网络;

`Y`: 网络输出;

`E`: 网络误差;

`Pf`: 最终输入延迟;

`Af`: 最终层延迟;

`tr`: 训练记录 (步数和性能)。

注意: 参数 `T` 是可选的, 并且只用于必须指明网络目标的场合。同样, `Pi` 和 `Pf` 也是可选的, 它们只用于存在输入延迟和层延迟的网络。

【例 10-5】 对一个神经网络进行自适应训练。

```
>> clear all;
p1 = [-1 0 1 0 1 1 -1 0 -1 1 0 1];
t1 = [-1 -1 1 1 1 2 0 -1 -1 0 1 1];
net = linearlayer([0 1],0.5);
[net,y,e,pf] = adapt(net,p1,t1);           %自适应训练
while (mae(e)<1e-20)
    [net,y,e,pf] = adapt(net,p1,t1);
end
mae(e)                                     %平均绝对误差
```

运行程序, 输出如下:

```
ans =
    0.4196
```

4. revert 函数

该函数用于将更新后的权值和阈值恢复到最后一次初始化的值, 其调用格式为:

```
net = revert (net)
```

如果网络结构已经发生了变化, 也就是说, 如果网络的权值和阈值之间的连接关系, 以及输入、每层的长度都与原来的网络结构有所不同, 那么该函数无法将权值和阈值恢复到原来的值。在这种情况下, 函数将权值和阈值都设置为 0。

【例 10-6】 对所创建的网络求其权值与阈值。

```
>> clear all;
net = newp([0 1; -2 2],1);
disp('默认网络的权值与阈值为: ')
net.iw{1,1}, net.b{1}
%改变网络的权值与阈值
net.iw{1,1} = [1 2];
net.b{1} = 5;
```




```
disp('显示改变网络的权值与阈值为: ')
net.iw{1,1}, net.b{1}
net = revert(net);           %更新网络权值与阈值
disp('更新网络权值与阈值为: ')
net.iw{1,1}, net.b{1}
```

运行程序，输出如下：

默认网络的权值与阈值为：

```
ans =
    0    0
ans =
    0
```

显示改变网络的权值与阈值为：

```
ans =
    1    2
ans =
    5
```

更新网络权值与阈值为：

```
ans =
    0    0
ans =
    0
```

10.4 神经网络初始函数

在 MATLAB 神经网络工具箱中提供了若干函数实现网络的初始化，下面分别给予介绍。

1. init 函数

该函数用于对神经网络进行初始化，其调用格式为：

```
net = init(Net)
```

其中，

Net：为待初始化的神经网络；

net：为返回参数，表示已经初始化后的神经网络。

Net 为 net 经过一定的初始化修正而成。修正后，前者的权值和阈值都发生了改变。

【例 10-7】 求创建网络对应的权值与阈值。

```
>> clear all;
net = newp([0 1; -2 2],1);
disp('默认网络的权值与阈值为: ')
net.iw{1,1}
```



```
net.b{1}
P = [0 1 0 1; 0 0 1 1];
T = [0 0 0 1];
net = train(net,P,T);
disp('网络经过训练后的权值与阈值为: ')
net.iw{1,1}
net.b{1}
disp('初始化修正后网络的权值与阈值为: ')
net = init(net);
net.iw{1,1}
net.b{1}
```

运行程序，输出如下：

默认网络的权值与阈值为：

```
ans =
     0     0
ans =
     0
```

网络经过训练后的权值与阈值为：

```
ans =
     1     2
ans =
    -3
```

初始化修正后网络的权值与阈值为：

```
ans =
     0     0
ans =
     0
```

2. initlay 函数

该函数特别适用于层结构的神经网络的初始化，其调用格式为：

```
net = initlay(Net)
```

其中，

Net 为待初始化的神经网络；

net 为初始化后的神经网络

info = initlay('code')依据 code 值的不同，返回有关函数的信息，包括：

- 当 code=fpnames 时返回函数参数的名称。
- 当 code=fpdefaults 时返回默认的函数参数。

通过指定神经网络每一层的初始化函数 Net.layers{i}来调用该函数，初始化后的神经网络每一层都得到修正。



3. initnw 函数

该函数是一个层初始化函数，它按照 Nguyen-Widrow 准则对某层的权值和阈值进行初始化，其调用格式为：

```
net = initnw(Net,i)
```

其中，

Net: 待初始化的神经网络；

i: 层次索引；

net: 初始化后的神经网络。

4. initwb 函数

该函数也是一个层初始化函数，它按照设定每层的初始化函数对每层的权值和阈值进行初始化，其调用格式为：

```
net=initwb(Net,i)
```

其中，

Net: 待初始化的神经网络；

i: 层次索引；

net: 初始化后的神经网络。

10.5 神经网络输入函数

在 MATLAB 神经网络工具箱中提供了若干函数，用于实现神经网络的输入，下面分别给予介绍。

1. netsum 函数

该函数是一个输入求和函数，它通过将某一层的加权输入和阈值相加作为该层的输入，其调用格式为：

```
N = netsum({Z1,Z2,...,Zn},FP)
```

其中，

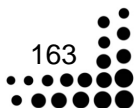
Zi(i=1,2,3,...): 第 i 个输入，它的数目可以是任意个；

FP: 行单元格数组函数的参数（忽略）；

dN_dZj = netsum('dz',j,{Z1,...,Zn},N,FP) 返回的是 netsum 的微分函数 dnetsum。

info = netsum('code') 依据 code 值的不同，返回有关函数的信息，包括：

- 当 code=names 时返回训练参数的名称；
- 当 code=type 时返回训练函数类型。
- 当 code= pnames 时表示返回函数全称；
- 当 code=pdefaults 时返回训练参数的默认值。





- 当 `code=fpcheck` 时返回一个错误非法函数参数。
- 当 `code=fullderiv` 时返回 0 或 1。

2. netprod 函数

与 `netsum` 的计算框架类似，不过该函数是输入求积函数，它将某一层的权值和阈值相乘作为该层的输入，其调用格式为：

```
N = netprod({Z1,Z2,...,Zn})
```

其中，

$Z_i(i=1,2,3,...)$ ：第 i 个输入，它的数目可以是任意个；

`dN_dZj = netprod('dn_dzj',j,{Z1,...Zn})` 返回的是 `netprod` 的微分函数 `dnetprod`。

`info = netprod('code')` 依据 `code` 值的不同，返回有关函数的信息，包括：

- 当 `code=deriv` 时返回导数函数名称。
- 当 `code=fullderiv` 时返回导数的次数。
- 当 `code=names` 时返回训练参数的名称；
- 当 `code=fpnames` 时返回函数参数的名称。
- 当 `code=fpdefaults` 时返回默认的函数参数。

3. concur 函数

该函数的作用在于使得本来不一致的权值向量和阈值向量的结构一致，以便于进行相加或相乘运算，其调用格式为：

```
concur(B,Q)
```

其中，

B： $N1 \times 1$ 维的权值向量；

Q：要达到一致化所需要的长度。

【例 10-8】 有两个加权输入向量 $Z1$ 和 $Z2$ ，试求其加权和与加权积。

```
>> clear all;
z1 = [1 2 4; 3 4 1];
z2 = [-1 2 2; -5 -6 1];
b = [0; -1];
c=concur(b,3)
s1= netsum(z1,z2)
s2= netsum(z1,z2,c)
p1=netprod(z1,z2)
p2=netprod(z1,z2,c)
```

运行程序，输出如下：

```
c =
     0     0     0
    -1    -1    -1
s1 =
```



```

    0    4    6
   -2   -2    2
s2 =
    0    4    6
   -3   -3    1
p1 =
   -1    4    8
  -15  -24    1
p2 =
    0    0    0
   15   24   -1

```

由此可看出, `netsum` 和 `netprod` 的运算规则就是将权值和阈值向量中对应的元素相加或相乘。同时也显示了函数 `concur` 的运行机理, 即修正后的矩阵就是由向量的副本组合而成的。

10.6 神经网络的传递函数

传递函数的作用是将神经网络的输入转换为输出。在 MATLAB 神经网络工具箱中提供了 `hardlim` 函数与 `hardlims` 函数实现神经网络的传递。下面分别对这两个函数进行介绍。

1. `hardlim` 函数

该函数为硬限幅传输函数。硬限幅传输函数可通过计算网络的输入得到该层的输出, 如果网络的输入达到门限, 则硬限幅传输函数的输出为 1; 否则, 为 0。这表明神经元可用来做出判断或分类, 其调用格式为:

在给定网络的输入向量矩阵 `N` 时, 返回该层的输出向量矩阵 `A`, 当 `N` 中的元素大于等于零时, 返回的值为 1, 否则为 0; `FP` 为性能参数 (可忽略)。

`dA_dN = hardlim('dn',N,A,FP)`: 返回 `A` 关于 `N` 的导数。如果 `A` 或 `FP` 为空, 则 `FP` 为默认参数, `A` 的值依据 `N` 来计算。

`info = hardlim('code')`: 依据 `code` 值的不同, 返回不同的信息。具体返回内容为:

- 当 `code=name` 时返回传输函数的全称。
- 当 `code=output` 时返回包含传输函数输出范围最小、最大值的二元向量。
- 当 `code=active` 时返回包含传输函数输入范围最小值、最大值的二元向量。
- 当 `code=fullderiv` 时依据 `N` 返回 1 或 0。
- 当 `code=fpnames` 时返回函数参数的名称。
- 当 `code=fpdefaults` 时返回函数默认参数。

该函数的原型函数为:

$$\text{hardlim}(n) = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$

【例 10-9】 利用 MATLAB 给出一个数组, 调用函数 `hardlim` 与 `hardlims` 对其进行分类。MATLAB 代码如下。

```
>>clear all;
%以 0.1 为步长，建立一个数组
N=-6:0.1:6;
A1=hardlim(N);
A2=hardlims(N);
plot(N,A1,'ro');
hold on
plot(N,A2,'b+');
hold on
```

运行程序，效果如图 10-4 所示。

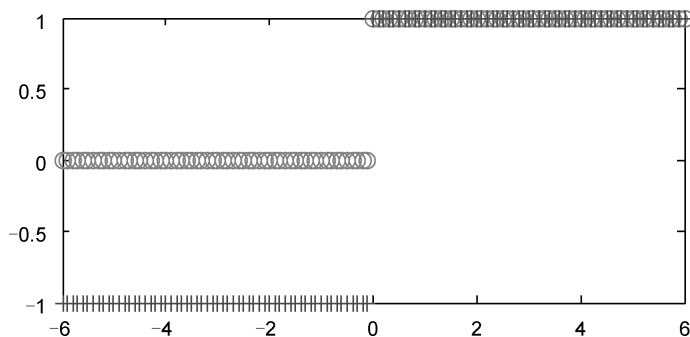


图 10-4 利用传递函数进行分类的结果

图中加号部分是函数 `hardlims` 的分类结果；圆圈部分是 `hardlim` 的分类结果。可以看出两个函数都成功地对数组进行了分类。

2. `hardlims` 函数

该函数为对称硬限幅传输函数。对称硬限幅传输函数可通过计算网络的输入得到该层的输出，如果网络的输入达到门限，则硬限幅传输函数的输出为 1；否则为-1。该函数的调用格式为：

```
A = hardlims(N,FP)
dA_dN = hardlims('dn',N,A,FP)
info = hardlims('code')
```

该函数的参数含义与 `hardlim` 函数的参数含义相同。

该函数的原型函数为：

$$\text{hardlims}(n) = \begin{cases} 1 & n \geq 0 \\ -1 & n < 0 \end{cases}$$

由此可见，以上两个函数可以实现神经网络的分类和判断功能。

【例 10-10】 用 MATLAB 写如图 10-5 所示的三层神经网络每层的输出表达式。

已知：P=[0.1 0.5; 0.3 -0.2]; S1=2; S2=3; S3=5。

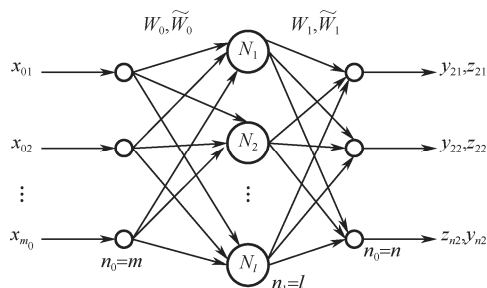


图 10-5 三层多输入多输出人工神经网络结构

其实现的 MATLAB 程序代码如下：

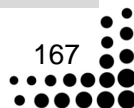
```
>> clear all;
P=[0.1 0.5;0.3 -0.2];           %已知输入向量数据
S1=2;S2=3;S3=5;                 %已知各层节点数
[r,q]=size(P);                   %求出输入向量的行和列
[W1,B1]=rands(S1,r);             %给出第一隐含层权值(-1,1)之间的随机值
[W2,B2]=rands(S2,S1);           %给出第二隐含层权值(-1,1)之间的随机值
[W3,B3]=rands(S3,S2);           %给出输出层权值(-1,1)之间的随机值
B10=cat(2,B1,B1);               %通过 B10=[B1,B1]将 S1×1 的 B1 矩阵合成为一个
                                %S1×q 的 B10 矩阵以便进行下面的加法运算
n1=W1*P+B10;                    %计算第一层的加权输入和
A1=hardlims(n1)                 %计算第一层输出表达式
B20=cat(2,B2,B2);              %用两个 S2×1 的 B2 矩阵组成一个 S2×q 矩阵
n2=W2*A1+B20;                  %计算第二层的加权输入和
A2=logsig(n2)                   %计算第二层输出表达式
B30=cat(2,B3,B3);              %用两个 S3×1 的 B3 矩阵组成一个 S3×q 矩阵
n3=W3*A2+B30;                  %计算输出层的加权输入和
A3=purelin(n3)                  %计算输出层输出表达式
```

运行程序，输出如下：

```
A1 =
    -1    -1
    -1    -1

A2 =
    0.2595    0.2595
    0.8480    0.8480
    0.2601    0.2601

A3 =
   -2.0291   -2.0291
   -1.2787   -1.2787
    0.9143    0.9143
    1.2701    1.2701
    0.1211    0.1211
```





10.7 神经网络求点积函数

在 MATLAB 神经网络工具箱中提供了 `dotprod` 函数，用于对权值求点积，它求得权值与输入之间的点积作为加权输入，其调用格式为：

```
Z = dotprod(W,P,FP)
dim = dotprod('size',S,R,FP)
dp = dotprod('dp',W,P,Z,FP)
dw = dotprod('dw',W,P,Z,FP)
info = dotprod('code')
```

其中， W 为 $S \times R$ 维的权值矩阵； P 为 Q 组 R 维的输入向量； FP 为性能参数（可忽略）； S 为层的维数； R 为输入的维数； Z 为 Q 组 S 维的 W 与 P 的点积；在函数返回值中， Z 为 $S \times Q$ 维的 W 与 P 的点积； dim 为权值大小； dp 为 Z 对 P 的导数； dw 为 Z 对 W 的导数。

`info = dotprod('code')`，依据 `code` 值的不同，返回有关函数的信息，包括：

- 当 `code=name` 时表示返回传输函数的全称。
- 当 `code=deriv` 时返回导数函数名称。
- 当 `code=wfullderv` 时返回权值矩阵的导数次数。
- 当 `code=fullderv` 时返回输入矩阵的导数次数。
- 当 `code=fpnames` 时返回函数参数的名称。
- 当 `code=fpdefaults` 时返回默认的函数参数。

【例 10-11】 建立两个矩阵（或向量），求其权值的点积。

```
>> clear all;
W = rand(4,3)           % 创建一个 4×3 的矩阵，所有元素都小于 1
P = rand(3,1)           % 创建一个 3×1 的矩阵，所有元素都小于 1
Z = dotprod(W,P)
```

运行程序，输出如下：

```
W =
    0.0046    0.0844    0.4314
    0.7749    0.3998    0.9106
    0.8173    0.2599    0.1818
    0.8687    0.8001    0.2638

P =
    0.1455
    0.1361
    0.8693

Z =
    0.3872
    0.9588
    0.3124
    0.4646
```


第 11 章 BM 网络与 BSB 网络

算法分析与实现

BM 网络的算法根据其用途可以分为工作规则和学习规则，本章将对 BM 网络进行研究。

11.1 Boltzmann 神经网络

Hinton 等人在 1985 年将模拟退火算法引入神经网络中，提出了 Boltzmann 机网络，简称 BM 网络。

11.1.1 BM 网络的基本结构

BM 网络结构与离散 Hopfield 网络结构相似，由 N 个神经元组成，每个神经元取 0 或 1 二值输出，且神经元之间以对称连接权相互连接。与 Hopfield 网络不同的是，BM 网络通常将整个神经元分为可视层和隐含层两大部分。可视层又可分为输入部分和输出部分，但与一般阶层网络的区别在于它没有明显的层次界限，且神经元之间不是单向连接而是双向连接的，如图 11-1 所示。

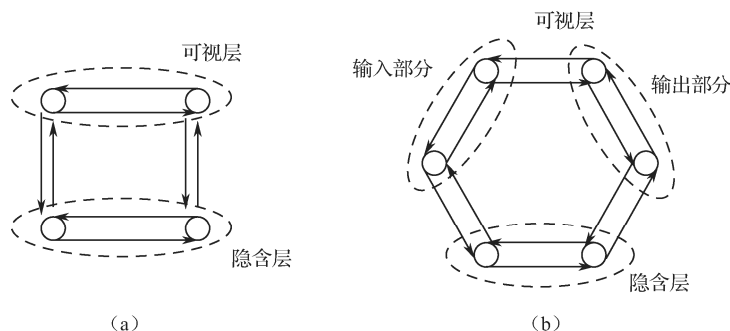


图 11-1 BM 网络结构

11.1.2 BM 模型的学习

BM 网络的算法根据其用途可以分为工作规则和学习规则。工作规则也就是网络的状态



更新规则，主要用于优化组合问题。学习规则就是连接权值和输出阈值的修正规则，主要用于将网络作为一个外界概率分布模拟机的场合，这也是 BM 网络的一个独特的方面。首先介绍 BM 网络的工作规则。

实际上，BM 网络的工作规则就是模拟退火器算法的具体体现，其步骤如下。

假定网络有 N 个神经元，各神经元之间的连接权值为 w_{ij} ，各神经元的输出阈值为 θ_i ，输出为 u_i ，神经元 i 的内部状态为 H_i ， $i, j=1, 2, \dots, N$ 。网络温度 $T|_{t=0}=T_0$ ，为 w_{ij} 和 θ_i 赋予区间 $[-1, 1]$ 之间的随机值，并令 $w_{ij} = w_{ji}$ 。

- (1) 从 N 个神经元中随机选取一个神经元 i 。
- (2) 按照式 (11-1) 求出神经元 i 的输入总和，即内部状态 H_i 。

$$H_i(t) = \sum_{j=1, j \neq i}^N w_{ij} u_j(t) - \theta_i \quad (11-1)$$

- (3) 按照式 (11-2) 得到的概率将神经元状态更新为 1。

$$P[u_i(t+1)=1] = \frac{1}{1 + \exp(-H_i(t)/T)} \quad (11-2)$$

- (4) i 以外的神经元的输出状态保持不变。

$$u_j(t+1) = u_j(t) \quad j=1, 2, \dots, N, \quad j \neq i \quad (11-3)$$

- (5) 令 $t=t+1$ ，按照式 (11-4) 计算新的温度参数 $T(t+1)$ 。

$$T(t+1) = \frac{T_0}{\log(t+1)} \quad (11-4)$$

- (6) 返回步骤 (1)，直到温度参数 T 小于预先设定的截止温度 T_d 。

在步骤 (3) 中更新网络状态，一般有两种方法。

(1) 当 $H_i(t) > 0$ 时，直接令 $u_i(t+1)=1$ ；当 $H_i(t) < 0$ 时，产生一个位于区间 $[0, 0.5]$ 内的随机数 $\varepsilon(t)$ ；当 $P[u_i(t+1)=1] > \varepsilon$ 时，令 $u_i(t+1)=1$ ，否则令 $u_i(t+1)=u_i(t)$ 。

(2) 当 $H_i(t) > 0$ 时，直接令 $u_i(t+1)=1$ ；当 $H_i(t) < 0$ 时，当 $P[u_i(t+1)=1]$ 大于预先设定的概率值 $\varepsilon(\varepsilon < 0.5)$ 时，令 $u_i(t+1)=1$ ，否则令 $u_i(t+1)=u_i(t)$ 。

此外，关于初始温度 T_0 和结束温度 T_e ，一般凭经验给出。

BM 网络除了可以解决优化组合问题外，还可以通过网络训练模拟外界给出的概率分布，实现概率意义下的联想记忆。联想记忆分为自联想记忆和互联想记忆两种模式。当把一组记忆模式及其概率分布函数提供给 BM 网络的可视层后，网络按照一定的学习规则进行学习，学习结束后，当网络状态按照工作规则进行不断转移时，网络的各个状态间按照记忆的学习模式的概率分布出现，即概率大的状态出现的频率高，概率小的状态出现的频率低。这种概率意义下的联想就称为自联想记忆。

如果将某个记忆模式提供给网络的输入部分，同时，在输出部分按照给定的概率分布给出一组目标输出模式。此时给出的概率分布函数实际上是输出模式相对于输入模式的条件概率分布。BM 网络正是通过记忆这种条件概率分布函数来完成互联想记忆的，例如，由 BM 网络对柴油机进行故障诊断，当为网络提供一个“排气筒有黑烟”的故障模式后，在网络的输出部分按产生这种故障现象的原因的概率大小提供一系列的输出模式，如汽缸点火位置不对，油料中有杂质等。这样就构成了网络的学习模式样本。



无论自联想记忆还是互联想记忆,其实质都是通过学习目标概率分布函数,将其记忆并在以后的回想过程中将这一概率分布再现出来。

下面首先介绍自联想记忆学习规则。假设网络共有 N 个神经元,其中可视层有 n 个神经元,隐含层有 $m = N - n$ 个神经元。可视层有 $\rho = 2^n$ 个状态,隐含层有 $q = 2^m$ 个状态,整个网络则有 $M = 2^N$ 个状态。各层的状态可表示为:可视层状态 $U_a = (u_1, u_2, \dots, u_n)$, 隐含层状态 $U_b = (u_1, u_2, \dots, u_m)$, 其中 $a = 1, 2, \dots, p$, $b = 1, 2, \dots, q$ 。整个网络的状态概率分布函数为 $Q(U_a, U_b)$, 网络的连接权值和输出阈值分别为 w_{ij} 和 θ_j , $i, j = 1, 2, \dots, N$, 网络在第 k 个状态时的能量为 $E_K(U_a, U_b)$, 学习过程如下。

(1) 初始化。将连接权 w_{ij} 赋予区间 $[-1, 1]$ 之间的随机值并令 $\theta_j = 0$ 。

(2) 按给定的外界概率(目标概率分布) $P(U_a)$ 将网络可视层的各神经元固定在某一状态 $U_a = (u_1, u_2, \dots, u_n)$ 。

(3) 从温度 T_0 开始,按照网络工作规则(即模拟退火算法)对网络隐含层的各神经元的输出进行状态更新,直到达到温度 T_d 下的平衡状态 $U_b = (u_1, u_2, \dots, u_m)$ 。

(4) 在隐含层的平衡状态下,保持温度 T_d 不变,再进行 L 次全网络的状态更新,每次更新后,当神经元 i 和 j 同时为 1 时,计算式(11-5)(学习过程):

$$n_{ij}^+ = n_{ij}^+ + 1 \quad (11-5)$$

(5) 重新从温度 T_0 开始,按照网络工作规则对全网络神经元状态 u_i 进行更新,直到达到温度 T_d 下的平衡状态 $U = (u_1, u_2, \dots, u_n, u_{n+1}, u_{n+2}, \dots, u_{n+m})$, $n + m \leq N$ 。

(6) 在网络的平衡状态下,保持温度 T_d 不变,再进行 L 次全网络的状态更新,每次更新后,当神经元 i 和 j 同时为 1 时,计算式(11-6)(反学习过程):

$$n_{ij}^- = n_{ij}^- + 1 \quad (11-6)$$

(7) 返回步骤(2),一共进行 M 次循环,并要求 $M > p$, p 为可视层的状态个数。

(8) 计算对称概率 P_{ij}^+ , P_{ij}^- :

$$P_{ij}^+ = \frac{1}{M \times L} n_{ij}^+, \quad P_{ij}^- = \frac{1}{M \times L} n_{ij}^- \quad i, j = 1, 2, \dots, N \quad (11-7)$$

(9) 按照式(11-8)调整网络的连接权值 w_{ij} :

$$w_{ij} = w_{ij} + \varepsilon(P_{ij}^+ - P_{ij}^-) / T_d \quad i, j = 1, 2, \dots, N \quad (11-8)$$

(10) 返回步骤(2),直到循环次数大于或等于预先设定的值。

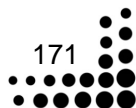
下面介绍互联想记忆学习准则。在以前假设条件的基础上,进一步假设可视层输入部分有 n_i 个神经元,对应 $p_i = 2^{n_i}$ 个状态 $U_a = (u_1, u_2, \dots, u_{n_i})$; 输出部分有 n_o 个神经元,对应 $p_o = 2^{n_o}$ 个状态 $L_c = (l_1, l_2, \dots, l_{n_o})$ 。

目标联合概率分布 $P(U_a, L_c) = P(U_a)P(L_c | U_a)$, 学习步骤如下。

(1) 初始化。将连接权 w_{ij} 赋予区间 $[-1, 1]$ 之间的随机值并令 $\theta_j = 0$ 。

(2) 随机选取输入模式 $U_a = (u_1, u_2, \dots, u_{n_i})$ 提供给可视层的输入部分。

(3) 按照目标条件概率分布 $P(L_c | U_a)$, 将输出模式 $L_c = (l_1, l_2, \dots, l_{n_o})$ 固定在可视层的输





出部分上。

(4) 从温度 T_0 开始, 按照网络工作规则 (即模拟退火算法) 对网络隐含层的各神经元的输出进行状态更新, 直到达到温度 T_d 下的平衡状态。

(5) 在隐含层的平衡状态下, 保持温度 T_d 不变, 再进行 L 次全网络的状态更新, 每次更新后, 当神经元 i 和 j 同时为 1 时, 计算式 (11-9) (学习过程):

$$n_{ij}^+ = n_{ij}^+ + 1 \quad (11-9)$$

(6) 重新从温度 T_0 开始, 按照网络工作规则对全网络神经元状态 u_i 进行更新, 直到达到温度 T_d 下的平衡状态。

(7) 在网络的平衡状态下, 保持温度 T_d 不变, 再进行 L 次全网络的状态更新, 每次更新后, 当神经元 i 和 j 同时为 1 时, 计算式 (11-10) (反学习过程):

$$n_{ij}^- = n_{ij}^- + 1 \quad (11-10)$$

(8) 返回步骤 (3), 一共进行 M_1 次循环, 并要求 $M_1 > p_0$, p_0 为可视层输出部分的状态个数。

(9) 计算对称概率 P_{ij}^+ , P_{ij}^- :

$$P_{ij}^+ = \frac{1}{M \times L} n_{ij}^+, \quad P_{ij}^- = \frac{1}{M \times L} n_{ij}^- \quad i, j = 1, 2, \dots, N \quad (11-11)$$

(10) 按照式 (11-12) 调整网络的连接权值 w_{ij} :

$$w_{ij} = w_{ij} + \varepsilon(P_{ij}^+ - P_{ij}^-) / T_d \quad i, j = 1, 2, \dots, N \quad (11-12)$$

(11) 返回步骤 (2), 选取下一组学习模式进行 M_2 次循环, 且要求 $M_2 > p_i$, p_i 为可视层输入部分的状态个数。

(12) 从步骤 (2) ~ (11) 进行 Y 次循环后学习结束, 其中 Y 为预先设定的最大循环次数。

在学习结束后网络进行回想时, 当给网络的输入部分提供一输入模式 U_a 后, 对网络输入部分按照网络工作规则进行状态更新, 在网络的输出部分各个状态出现的概率将符合学习的希望概率分布 $P(L_c | U_a)$ 。

11.1.3 BM 网络的实现

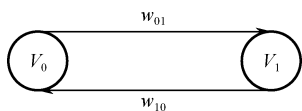


图 11-2 含有两个神经元的 BM 网络结构

这里利用一个非常简单的例子来说明 BM 网络的自联想学习规则, 设网络只有两个神经元, 一个为可视层神经元 V_0 , 另一个为隐含层神经元 V_1 。两个神经元各有 0, 1 两种状态, 可视层输入为 1 的概率为 0.1, 网络结构如图 11-2 所示。

MATLAB 神经网络工具箱中没有关于 BM 网络的函数, 因此, 需要利用 MATLAB 的数学计算功能来实现 BM 网络的学习和训练。

首先赋予网络权值 $w_{10} = -10$, $w_{01} = -10$ 。设网络温度为 $T_0 = 100$, $T_d = 10$, 以快速降温的方式, 按照网络工作规则对网络进行状态更新, 以得到学习和训练之前的状态概率分布, 即



网络状态的初始分布情况，如表 11-1 所示。

网络状态对应的能量按照式 (11-13) 计算：

$$E_i = -\frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^N w_{ij} u_i u_j + \theta_i u_i \quad (11-13)$$

表 11-1 网络状态的初始分布

网 络 状 态		对 应 能 量	状态出现的理论概率	状态出现的实际概率
可 视 层	隐 含 层			
0	0	0.0000	0.0000	0.1310
0	1	-8.0000	0.0000	0.2530
1	0	-10.0000	0.0000	0.3580
1	1	-8.0000	0.0000	0.2580

其中， N 为网络状态的个数，这里 $N=4$ 。

网络各个状态出现的理论概率按照式 (11-14) 计算：

$$Q(E_i) = \frac{\exp(-E_i / T)}{Z}, \quad Z = \sum_{i=1}^N \exp(-E_i / T) \quad (11-14)$$

由表 11-1 可得网络的可视层输出是 1 的概率为 $0.2580+0.2580=0.6160$ 。

接下来进行网络学习，已知 $P(U_a) = P(U_0 = 1) = 1$ 。设 $T_0 = 100$, $T_d = 10$, $L=1000$, $M = 128$, $Y = 6$, $\varepsilon = 0.1$ ，如表 11-2 所示为每次大循环后，网络可视层输出为 1 的概率分布。

表 11-2 网络可视层输出为 1 的概率分布

1	2	3	4	5	6
0.3190	0.3190	0.2260	0.2260	0.1590	0.1390

网络学习结束后，以网络温度 $T_0 = 100$, $T_d = 10$ 的快速降温方式，按照网络工作规则，令网络进行回想。网络连接权的初始值为 $w_{10} = -16.7656$, $w_{01} = -16.7656$ ，网络的回想结果如表 11-3 所示。

表 11-3 网络回想结果

网 络 状 态		对 应 能 量	状态出现的理论概率	状态出现的实际概率
可 视 层	隐 含 层			
0	0	0.000	0.4922	0.1460
0	1	-16.7656	0.4219	0.7560
1	0	7.8008	0.0156	0.0430
1	1	7.5806	0.0703	0.0550

由表 11-3 可得网络可视层输出为 1 的概率为 $0.0430+0.0550=0.0980$ 。可见网络的实际概率分布和希望概率分布 0.1 已经十分接近。



11.2 BSB 神经网络

盒中脑 (Brain-State-in-a-Box, BSB) 神经网络模型首先是由 Anderson 等人于 1977 年提出的, Golden 等人对该模型进行了深入的研究。BSB 模型是一种节点之间存在横向连接和节点自反馈的单层网络, 可用作自联想最邻近分类器, 并可存储任何模拟向量模式。

1. BSB 神经网络模型

BSB 网络模型可用如下方程描述:

$$X(k+1) = g(X(k) + \alpha W X(k)) \quad k = 0, 1, 2, \dots$$

初始条件为 $X(0) = X_0$, 其中 $X(k) = (x_1(k), x_2(k), \dots, x_n(k)) \in R^n$ 表示 k 时刻的状态向量, 参数 α 是一个正值, 用于控制层内反馈的大小。 $W \in R^{n \times n}$ 为对称的权矩阵, 传递函数 g 的第 i 个坐标通常为以下形式:

$$g_i(X) = \begin{cases} 1 & x_i \geq 1 \\ x_i & |x_i| < 1 \\ -1 & x_i \leq -1 \end{cases}$$

随着时间的推移, 每个状态 x_i 逐渐趋近于 ± 1 。实际上, 当系统达到某个平衡态后, 状态 (x_1, x_2, \dots, x_n) 进入由 $(\pm 1, \pm 1, \dots, \pm 1)$ 构成的盒子某一角。

2. BSB 网络的实现函数

神经网络工具箱中没有为 BSB 网络提供专门的函数工具, 因此, 无法利用神经网络工具箱中的函数创建、训练并应用网络。但是, Hugh Pasika 于 1997 年基于 MATLAB 平台开发了 BSB 网络的实现函数, 其 MATLAB 源程序代码如下。

```
function C=BSB(X,beta,multi)
%function C=BSB(X,beta)
%这个 M 文件为盒中脑模型示例
%X 为负输入矩阵
%beta 为负反馈系数
%C 为返回负反馈收敛迭代
%于 Hugh Pasika 1997 年开发
hold on
flag=0;
x=x(:);
c=2; %C is a general purpose counter
W=[.035 -.005; -.005 .035];
%set axes
set(gca,'XLim',[-1 1]);
set(gca,'XLim',[-1 1]);
%plot first point
```



```

plot(x(1),x(2),'ob');
Og=x';
%plot center lines
plot([0,0],[1,-1],'+');
plot([1,-1],[0,0],'+');
%label plot
set(gca,'YTick',[-1 1]);
set(gca,'XTick',[-1 1]);
while flag<1
    y=x+beta*W*x;
    x=(y(:,>-1)*(-1)+(y(:,>1)+(y(:,>-1 & y(:,>1)<1).*y;
    u(c,:)=x';
    c=c+1;
    if u(c-1,:)==u(c-2,:),
        flag=10;
        c=c-3;
    end
end
u=u(2:c+1,:);
Og
plot([Og(1,1) u(1,1)],[Og(1,2) u(1,2)],'-b')
plot(u(:,1),u(:,2),'ob')
plot(u(:,1),u(:,2),'-b')
drawnow
fprintf(1,'It took %g iteration for a stable point to be reached.\n\n',c);
set(gca,'Box','on')
hold off

```

读者可以将上述代码保存为一个名为 **BSB.m** 的文件，存储到 **MATLAB** 的根目录下，然后在 **MATLAB** 主窗口中将 **Current Directory** 当前目录修改为 **MATLAB** 的根目标，在命令窗口中输入“**help bsb**”后按回车键，出现如下的帮助信息。

```

help bsb
function C=BSB(X,beta)
    这个 M 文件为盒中脑模型示例
    X 为负输入矩阵
    beta 为负反馈系数
    C 为返回负反馈收敛迭代
    于 Hugh Pasika 1997 年开发

```

帮助信息解释了函数各个参数的意义，其中：

x 为输入向量；

beta 为反馈因子；

c 为最大迭代次数。



下面给出定期输入向量和反馈因子，演示 bsb 函数的功能。令 $x=[0.5; -0.6]$ ， $\beta=0.5$ ， $c=100$ ， c 用于限制迭代次数。迭代终止的充分条件是网络已经收敛或者网络的迭代次数达到最大值 c 。在命令行窗口中输入以下代码后按回车键，输出如下：

```
Og =  
    0.5000   -0.6000  
  
It took 133 iteration for a stable point to be reached.  
x=[0.5; -0.6];  
beta=0.5;  
c=100;  
BSB(x,beta,c)  
ans=  
    25
```

其中 Og 表示网络的初始输入值为 $(0.5, -0.6)$ 。 $ans=25$ 表示网络经过 25 次迭代后，初始值就达到了箱子的一角 $(1, -1)$ ，因为初始值与它的距离最小。

第 12 章 感知器网络工具箱函数及其应用

感知器是一种前馈神经网络，是神经网络中的一种典型结构。感知器具有分层结构，信息从输入层进入网络，逐层向前传递到输出层。根据感知器神经元的变换函数、隐层数以及权值调整规则的不同，可以形成具有各种功能特点的神经网络。

感知器使用特征向量来表示前馈式人工神经网络，它是一种二元分类器，把矩阵上的输入 x （实数值向量）映射到输出值 $f(x)$ 上（一个二元的值）。 $f(x)$ 的形式为：

$$f(x) = \begin{cases} 1, & w \cdot x + b > 0 \\ 0, & \text{其他} \end{cases}$$

w 是实数的表式权重的向量， $w \cdot x$ 是点积； b 是偏置，一个常数不依赖于任何输入值。偏置可以认为是激励函数的偏移量，或者给神经元一个基础活跃等级。

$f(x)$ （0 或 1）用于对 x 进行分类，看它是肯定的还是否定的，这属于二元分类问题。如果 b 是否定的，那么加权后的输入必须产生一个肯定的值并且大于 $-b$ ，这样才能令分类神经元大于阈值 0。从空间上看，偏置改变了决策边界的位置（虽然不是定向的）。

由于输入直接经过权重关系转换为输出，所以感知器可以被视为最简单形式的前馈式人工神经网络。

MATLAB 神经网络提供了大量的与神经网络感知器相关的函数。在 MATLAB 工作空间的命令行中输入“help perceptron”，便可得到与神经网络感知器相关的信息，进一步利用 help 命令又能得到相关函数的详细介绍。

12.1 创建函数

可通过感知器生成函数来创建一个感知器，并且可对感知器进行初始化、仿真和训练等。

在 MATLAB 神经网络工具箱中提供了 newp 函数，用于创建一个感知器网络，其调用格式为：

格式 net=newp(PR, S, TF, LF)

其中，

PR: $R \times 2$ 的输入向量最大值和最小值构成的矩阵；

S: 神经元的个数；

TF: 传输函数设置，为 hardlim 函数或者 hardlins 函数，默认为 hardlim 函数；

LF: 学习函数设置，为 learnp 函数或者 learnpn 函数，默认为 learnp 函数；

net: 生成的感知器网络。

【例 12-1】用 newp 函数设计一个单输入和一个神经元的感知器神经网络，输入的最小



值和最大值为[0 2]。

```
>> clear all;
P = [0 2];
T = [0 1];
net = newp(P,T);
%观察生成的神经网络
>> whos          %显示
Name      Size      Bytes  Class      Attributes
P         1x2         16   double
T         1x2         16   double
net       1x1      13496  network
net 的变量属性是网络对象的结构体，可以输入 net 后查看 net 结构体的相关信息。
>> inputweights = net.inputweights{1,1}
inputweights =
    Neural Network Weight
        delays: 0
      initFcn: 'initzero'
initSettings: (none)
        learn: true
      learnFcn: 'learnp'
    learnParam: (none)
         size: [1 1]
    weightFcn: 'dotprod'
    weightParam: (none)
      userdata: (your custom info)
```

从中可以看到，默认的学习函数为 `learnp`，后面会讨论。网络输入给 `hardlim` 传递函数的量为数量积 `dotprod`，即输入量和权值矩阵的乘积，然后再加上阈值。默认的初始化函数为 `initzero`，即权值的初始置为 0。

```
>> biases = net.biases{1}
biases =
    Neural Network Bias
      initFcn: 'initzero'
        learn: true
      learnFcn: 'learnp'
    learnParam: (none)
         size: 1
      userdata: (your custom info)
```

【例 12-2】 建立一个感知器，并对其目标向量进行仿真。

```
>> clear all;
%初始条件
```



```

PR=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1;
    0 1;0 1;0 1;0 1;0 1;0 1;0 1]; %取值范围
S=1;%神经元数目
%创建感知器神经网络
NET=newp(PR,S);
%训练样本
P=[1 1 1 0 1 1 0 1 1 0 1 1 1 1;
    1 1 0 0 1 0 0 1 1 0 1 0 1 1;
    1 1 1 1 0 1 0 1 1 1 1 0 1 1;
    1 1 1 1 1 1 0 1 1 0 0 1 1 1;
    0 1 0 1 1 0 1 1 0 1 1 1 0 1;
    1 1 1 1 1 0 0 1 1 0 0 1 1 1;
    0 1 1 1 1 0 1 1 1 1 0 1 1 1;
    1 1 1 1 0 1 0 1 0 0 1 0 0 1;
    1 1 1 1 0 1 1 1 1 1 0 1 1 1;
    1 1 1 1 0 1 1 1 1 0 1 1 1 0];
%目标函数
T=[1 0 1 0 1 0 1 0 1 0];
%训练感知器
[net,tr]=train(NET,P,T);
%感知器的权值
W=net.IW{1}
%感知器的阈值
B=net.b{1}
%每次训练的步长
epoch1=tr.epoch;
%每一步训练的误差
perf1=tr.perf;
%对训练后的感知器进行仿真
test=[1 1 0 1 0 1 1 0 1 1 0 1 1 1;
    1 1 0 0 1 0 0 1 1 0 1 1 1 1;
    0 1 0 1 1 0 1 1 0 1 1 1 0 1;
    1 1 1 1 1 0 0 1 1 0 0 1 1 1;]; %实验数据
result=sim(net,test)

```

运行程序，输出如下：

```

W =
    -2     0    -1     1    -2    -1     3    -1    -1     6     1     0    -1     0     0
B =
     0
result =
     1     0     1     0

```



12.2 显示函数

在 MATLAB 神经网络工具箱中提供了 `plotpv` 及 `plotpc` 函数，用于显示感知器网络。下面分别对这两个函数进行介绍。

1. `plotpv` 函数

该函数用于在感知器向量图中绘制分界线，其调用格式为：

```
plotpv(P,T)
plotpv(P,T,V)
```

其中，

P: Q 组 R 维的输入向量；

T: Q 组 S 维的双目标向量；

V=[x_min x_max y_min y_max]: 图形的最大值，绘制工作必须位于 **V** 所限定的范围中；

`plotpv(P,T)`: 以 **T** 为标尺，绘制 **P** 的列向量；

`plotpv(P,T,V)`: 在 **V** 的范围中绘制 **P** 的列向量。

2. `plotpc` 函数

该函数用于绘制感知器的输入向量的目标向量，其调用格式为：

```
plotpc(W,B)
plotpc(W,B,H)
```

其中，

W: $S \times R$ 维的加权矩阵 (R 必须小于等于 3)；

B: $S \times 1$ 维的阈值向量；

H: 最后画线的控制权；

`plotpc(W,B)`: 返回的是对所绘制分界线的控制权；

`plotpc(W,B,H)`: 用于在绘制新线之前检验最新绘制的分界线。

【例 12-3】 对给定某感知器的输入变量 **p** 和目标变量 **t** 绘制其样本点及类别，然后给定感知器的权值和阈值，绘制其分界线。

```
>> clear all;
p=[0 0 1 1;0 1 0 1];
t=[0 0 0 1];
%绘制输入向量和目标向量
plotpv(p,t)
net=newp(minmax(p),1);           %创建一个感知器网络
%设定权值
net.iw{1,1}=[-1.2 -0.5];
net.b{1}=1;                       %设定阈值
```



```
plotpc(net.iw{1,1},net.b{1})
title('向量类别');
```

运行程序，效果如图 12-1 所示。

【例 12-4】 假定已给出某感知器的输入变量 P 和目标变量 T ，绘制其曲线，然后给定感知器的权值和阈值，绘制其分界线。

```
>> clear all;
%定义输入向量及目标函数
P = [-0.5 -0.5 +0.3 -0.1 -0.8; ...
      -0.5 +0.5 -0.5 +1.0 +0.0];
T = [1 1 0 0 0];
plotpv(P,T);                                %绘制样本点
net = newp([-40 1; -1 50],1);                %创建感知器神经网络
hold on
linehandle=plotpc(net.IW{1},net.b{1});       %返回分界线句柄
net.adaptParam.passes = 3;
linehandle=plotpc(net.IW{1},net.b{1});
for a = 1:25
    [net,Y,E] = adapt(net,P,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);
drawnow;
end;
```

运行程序，效果如图 12-2 所示。

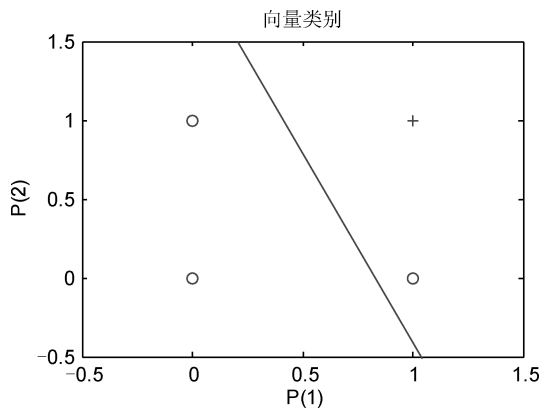


图 12-1 样本点类别及分界线效果图

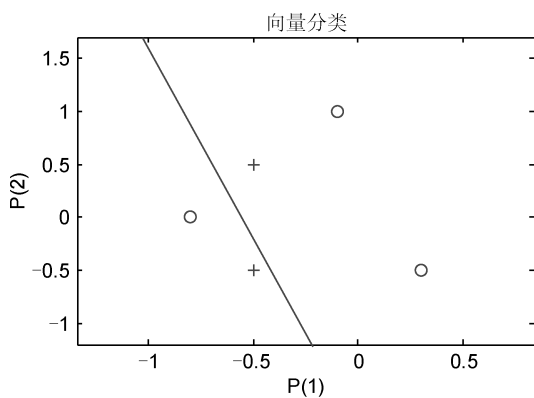


图 12-2 输入样本加网络初始分界线

12.3 性能函数

在 MATLAB 神经网络工具箱中提供了 `mae` 函数，用于实现感知器网络的性能，其调用格式为：



```

perf = mae(E,Y,X,FP)
dPerf_dy = mae('dy',E,Y,X,perf,FP)
dPerf_dx = mae('dx',E,Y,X,perf,FP)
info = mae('code')

```

其中，

E 为误差矩阵或向量 ($E=T-Y$ ，T 表示网络的目标向量)；

Y 为网络的输出向量（可忽略）；

X 为所有权值和偏值向量（可忽略）；

FP 为性能参数（可忽略）；

Perf 表示平均绝对误差；

dPerf_dy 表示返回 perf 对 Y 的导数；

dPerf_dx 表示返回 perf 对 X 的导数。

mae(code)将根据 code 值的不同，返回不同的信息，包括：

- 当 code=name 时表示返回函数全称；
- 当 code=pnames 时表示返回训练参数的名称；
- 当 code=pdefaults 时表示返回默认的训练参数。

【例 12-5】 创建一个感知器神经网络，并求其平均绝对误差。

```

>> clear all;
net = newp([-10 10],1);           %创建感知神经网络
p = [-10 -5 0 5 10];             %样本向量
t = [0 0 1 1 1];                 %目标向量
y = sim(net,p)                   %网络仿真
e = t-y                          %误差矩阵
perf = mae(e)                    %平均绝对误差

```

运行程序，输出如下：

```

y =
     1     1     1     1     1
e =
    -1    -1     0     0     0
perf =
    0.4000

```

【例 12-6】 感知器最重要的也是最实用的功能是对输入向量进行分类。本例尝试建立一个感知器模型，实现电路“或”门的功能，从而实现对输入的分类。

“或”门网络输入向量 P 和目标向量 T，其中， $P=[0\ 0\ 1\ 1;0\ 1\ 0\ 1]$ ； $T=[0\ 1\ 1\ 1]$ 。

```

>>clear all;
P=[0 0 1 1;0 1 0 1];
T=[0 1 1 1];
net=newp(minmax(P),1);
Y1=sim(net,P)

```



```
net.trainParam.epochs=25;
net=train(net,P,T);
Y2=sim(net,P)
perf=mae(Y2-T)
```

输出如下：

```
Y1 =
    1    1    1    1
TRAINC, Epoch 0/25
TRAINC, Epoch 4/25
TRAINC, Performance goal met.
Y2 =
    0    1    1    1
perf =
    0
```

由此可见，感知器在训练以前的输出是不符合要求的，经过 25 次训练后的输出已经和目标向量一致了，训练过程如图 12-3 所示。

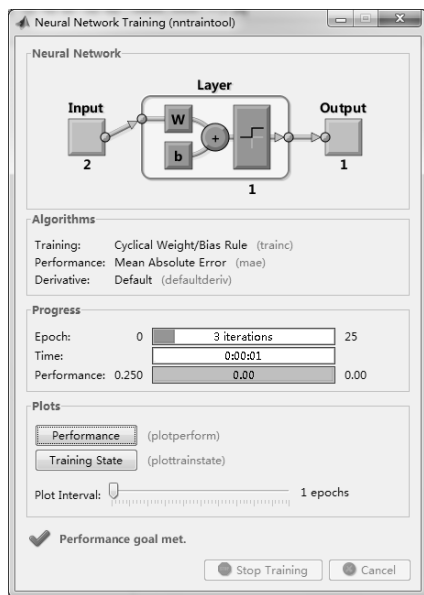


图 12-3 网络训练过程

本例创建的感知器只有一个神经元，这里也符合了神经网络的设计原则，即在设计神经网络时，在同样可以完成任务的情况下，尽是采用简单的结构。因为结构简单的神经网络计算负担轻，运行速度一般比较快。感知器的传递函数和学习函数都采用默认值，分别为 `hardlim` 和 `learnp`，这是因为网络的输出为 0, 1 的二值结构，只有采用 `hardlim` 才满足要求，输入向量中不存在奇异值，元素之间的距离也比较小，因此，采用 `learnp` 就足够了。

利用 `train` 函数对网络进行训练，训练的结果非常理想，训练后的网络成功实现了“或”功能。利用平均绝对误差函数 `mae` 计算网络的性能，结果为 0，从另一个方面说明了网络的

性能是非常好的。

【例 12-7】 建立一个感知器网络，对一组存在奇异值的输入向量 P 进行分类。

```
>> clear all;
P=[0 0.5 1.2 20;0 0.7 1.2 89];      %输入向量
T=[0 1 0 1];                        %目标向量
net=newp(minmax(P),1);               %创建感知器网络
net.trainParam.epochs=150;
net=train(net,P,T);                  %效果如图 12-4 所示
plotpv(P,T);                         %效果如图 12-5 所示
title('向量分类')
```

其中， T 表明了输入向量的分类情况。输入的向量分布如图 12-5 所示，其中的一个坐标点与其他点的距离都很大，而另外 3 个点相对比较集中，因此，输入向量中存在奇异值。

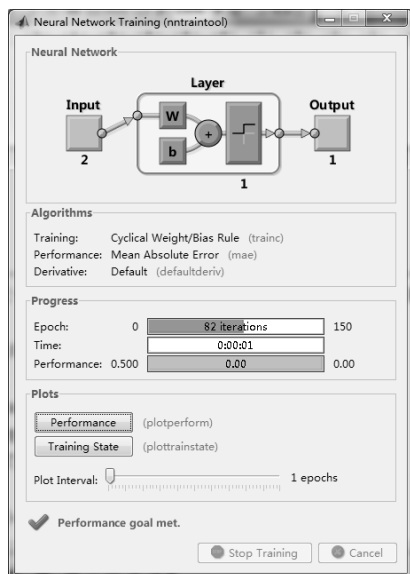


图 12-4 网络的训练过程

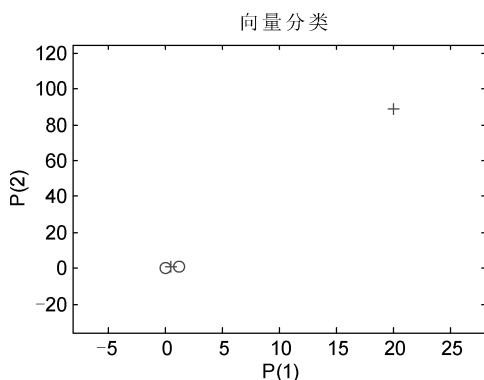


图 12-5 输入向量分布

根据需要，绘制分类线代码如下：

```
>> plotpc(net.iw{1},net.b{1});
```

得到训练的分类线如图 12-6 所示。

由于奇异值的存在，样本点相对比较集中，因此无法看出网络的分类性能。接下来将局部放大，检查网络的分类性能，代码为：

```
>> figure;
plotpv(P,T);
title('向量分类')
plotpc(net.iw{1},net.b{1});
%限制坐标起点和终点
axis([-2 2 -2 2]);
```




局部放大后的分类线如图 12-7 所示，两类样本点中分别有一个位于分类线上，由此可见，输入样本中的奇异点对于网络的分类性能影响很大。

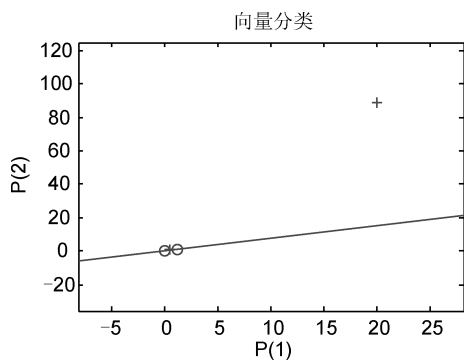


图 12-6 训练得到的分类线

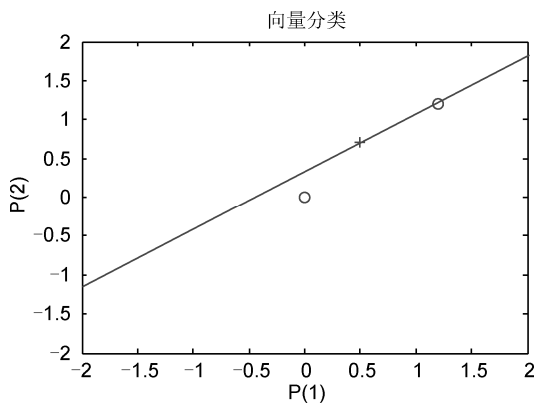


图 12-7 局部放大后的分类线

以上代码创建的感知器的学习函数为 `learnp`，接下来利用标准训练函数 `learnpn` 作为网络的学习函数。除感知器创建的代码有区别外，其余的全部一致，代码为：

```
>>net=newp(minmax(P),1,'hardlim','learnpn');
```

%创建感知器网络

网络的训练过程如图 12-8 所示。由图可见，网络经过 53 次学习后性能为 0，相对于图 12-4 中的 82 次，网络的训练时间有所减少。

图 12-9 是局部放大后的分类线。

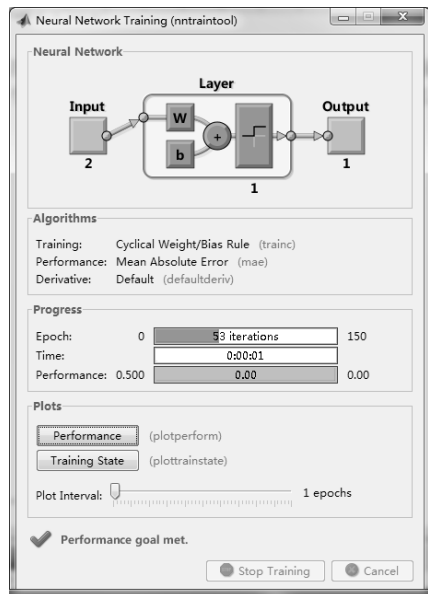


图 12-8 网络训练过程

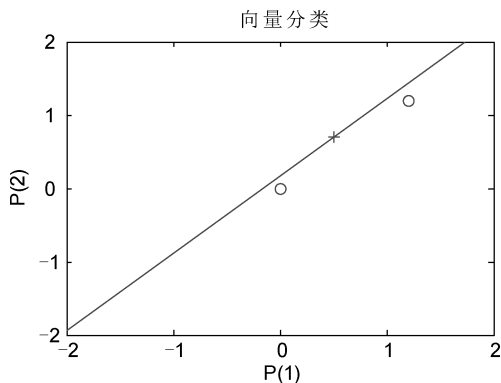


图 12-9 局部放大后的分类线

第 13 章 RBF 神经网络算法分析与应用

径向基（Radial Basis Function Neural Network, RBF）网络是一个三层的网络，除输入输出层之外仅有一个隐层。隐层中的转换函数是局部响应的高斯函数，而其他前向型网络，转换函数一般都是全局响应函数。由于这样的不同，要实现同样的功能，RBF 需要更多的神经元，这就是 RBF 神经网络不能取代标准前向型网络的原因。但是 RBF 的训练时间更短。它对函数的逼近是最优的，可以以任意精度逼近任意连续函数。隐层中的神经元越多，逼近越精确。

13.1 RBF 神经网络模型

RBF 函数网络是一种两层前向型神经网络，包含一个具有径向基函数神经元的隐层和一个具有线性神经元的输出层。

1. RBF 神经元模型

图 13-1 所示为一个有 R 个输入的径向基神经元模型。

径向基神经元的传递函数有各种各样的形式，但最常用的是高斯函数（radbas）。与前面介绍的神经元不同，神经元 radbas 的输入为输入向量 \mathbf{p} 和权值向量 \mathbf{w} 之间的距离乘以阈值 b 。径向基传递函数可表示为如下形式：

$$\text{radbas}(n) = e^{-n^2} \quad (13-1)$$

径向基神经元模型如图 13-1 所示。

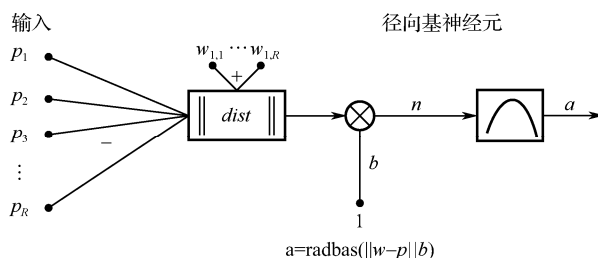


图 13-1 径向基神经元模型

径向基函数的图形如图 13-2 所示。

从图 13-2 中可以看出， n 为 0 时，径向基函数的输出最大值为 1，即权值的向量 \mathbf{w} 和输入向量 \mathbf{p} 之间距离减小时，输出就会增加。也就是说，径向基函数对输入信号在局部产生响应。函数的输入信号 n 靠近函数的中央范围时，隐层节点将产生较大的输出。由此可以看出



这种网络具有局部逼近能力，所以径向基函数网络也称为局部感知场网络。阈值 b 用于调整径向基神经元的敏感度，例如，假设神经元阈值为 $b=0.1$ ，那么对于任意与权值向量 w 之间距离为 8.33 的输入向量 p ，其输出都是 0.5。

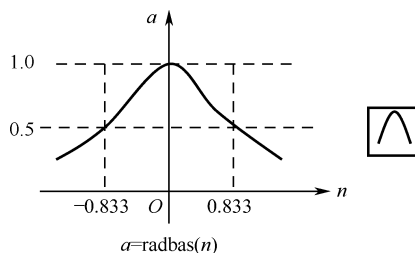


图 13-2 径向基函数

2. RBF 神经网络模型

RBF 神经网络同样是一种前馈反向传播网络，它有两个网络层：隐层为径向基层；输出为线性层，如图 13-3 所示。

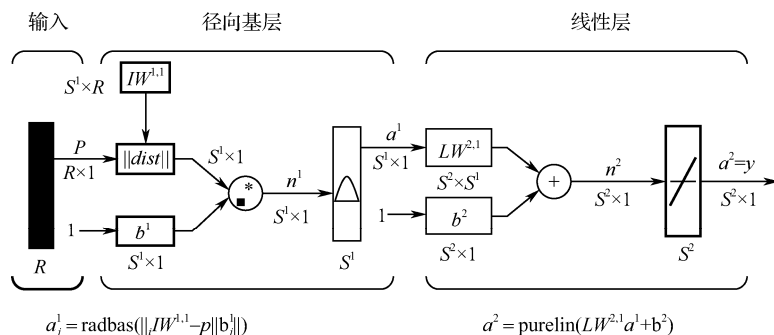


图 13-3 径向基函数网络的结构

网络的输出为：

$$a^2 = \text{purelin}(LW^{2,1}a^1 + b^2)$$

$$a^1 = \text{radbas}(n^1)$$

$$n^1 = \|IW - P\| b^1$$

$$= (\text{diag}((IW - \text{ones}(S^1, 1)P')(IW - \text{ones}(S^1, 1)P')))^{0.5} b^1$$

式中， $\text{diag}(x)$ 表示取矩阵向量主对角线上的元素组成的列向量；“ \wedge ”和“ \cdot ”分别表示数量乘方和数量乘积（即矩阵中各对应元素的乘方和乘积）。

由输入层、隐含层和输出层构成的一般 RBF 神经网络结构如图 13-4 所示。在 RBF 神经网络中，输入层仅仅起到传输信号的作用，与前面所讲述的神经网络相比较，输入层和隐含层之间可以看作连接权值为 1 的连接。输出层和隐含层所完成的任务是不同的，因而它们的学习策略也不相同。输出层是对线性权进行调整，采用的是线性优化策略。因而学习速度较快。而隐含层是对激活函数（格林函数或高斯函数，一般取高斯）的参数进行调整，采用的是非线性优化策略，因而学习速度较慢。

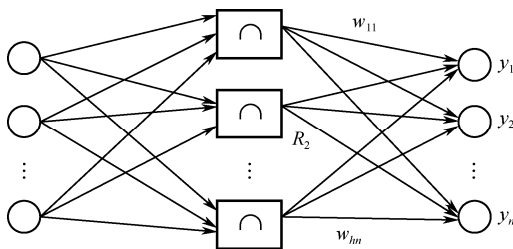


图 13-4 RBF 神经网络结构

可以从两方面理解径向基网络的工作原理。

(1) 从函数逼近的观点看：若把网络看成是对未知函数的逼近，则任何函数都可以表示成一组基函数的加权和。在径向基网络中，相当于选择各隐层神经元的传输函数，使之构成一组基函数逼近未知函数。

(2) 从模式识别的观点看：总可以将低维空间非线性可分的问题映射到高维空间，使其在高维空间线性可分。在径向基网络中，隐层的神经元数目一般比标准的 BP 网络要多，构成高维的隐单元空间，同时，隐层神经元的传输函数为非线性函数，从而完成从输入空间到隐单元空间的非线性变换。只要隐层神经元的数目足够多，就可以使输入模式在隐层的高维输出空间线性可分。在径向基网络中，输出层为线性层，完成对隐层空间模式的线性分类，即提供从隐单元空间到输出空间的一种线性变换。

13.2 RBF 神经网络的数学基础

13.2.1 内插问题

假定共有 N 个学习样本，其输入为 $s = (X_1, X_2, \dots, X_N)$ ，相应的样本输出，即教师信号（单输出）为 $t = (y_1, y_2, \dots, y_N)$ 。所谓的多变量内插问题是指寻找函数 F ，满足以下的内插条件：

$$y_i = F(X_i) \quad (13-2)$$

这是一个非常经典的数学问题，可以有多种解决方案，采用 RBF 神经网络也可以解决其学习问题。使用 RBF 神经网络前必须确定其隐节点的数据中心（包括数据中心的数目、值、扩展常数）及相应的一组权值。RBF 神经网络解决内插问题时，使用 N 个隐节点，并把所有的样本输入选为 RBF 神经网络的数据中心，且各基函数取相同的扩展常数。

现在再来确定网络的 N 个输出权值 $\mathbf{W} = (W_1, W_2, \dots, W_N)^T$ 。只要把所有的样本再输入一遍，便可解出各 w_i 的值。假定当输入 $X_i, i=1, 2, \dots, N$ 时，第 j 个隐节点的输出为：

$$h_{ij} = \Phi_j(\|X_i - c_j\|) \quad (13-3)$$

其中， $\Phi_j(\cdot)$ 为该隐节点的激活函数， $c_j = X_j$ 为该隐节点 RBF 函数的数据中心。于是可定义 RBF 神经网络的隐层输出阵为 $\mathbf{H} = [h_{ij}]$ ， $\mathbf{H} \in R^{N \times N}$ 。此时 RBF 神经网络的输出为：



$$f(X_i) = \sum_{j=1}^N h_{ij} w_j = \sum_{j=1}^N w_j \Phi_j(\|X_i - c_j\|) \quad (13-4)$$

令 $y_i = f(X_i)$, $\mathbf{y} = \mathbf{t}^T = (y_1, y_2, \dots, y_N)^T$, $\mathbf{W} = (w_1, w_2, \dots, w_N)^T$, 使得:

$$\mathbf{y} = \mathbf{H}\mathbf{W} \quad (13-5)$$

如果 $\mathbf{H} \in \mathbf{R}^{N \times N}$ 可逆, 即其列向量构成 \mathbf{R}^N 中的一组基, 则输出权向量可由式 (13-6) 得到:

$$\mathbf{W} = \mathbf{H}^{-1}\mathbf{y} \quad (13-6)$$

事实上, 根据 Micchelli 定理, 如果隐节点激活函数采用上述的径向基函数, 且 X_1, X_2, \dots, X_N 各不相同, 则隐层输出阵 \mathbf{H} 的可逆性是可以保证的。

因此, 如果把全部样本输入作为 RBF 网的数据中心, 网络在样本输入点的输出就等于教师信号, 此时网络对样本实现了完全内插, 即对所有样本误差为 0。但上述方案存在以下问题:

(1) 在通常情况下, 样本数据较多, 即 N 数值较大, 上述方案中隐层输出阵 \mathbf{H} 的条件数可能过大, 求逆时可能导致不稳定。

(2) 如果样本输出含有噪声, 此时由于存在过学习的问题, 做完全内插是不合适的, 而对数据逼近可能更合理。

为了解决这些问题, 可以采用下面的正则化网络。

13.2.2 正则化网络

假定 $S = \{(X_i, y_i) \in \mathbf{R}^n \times \mathbf{R} \mid i=1, 2, \dots, N\}$ 为我们想用函数 F 逼近的一组数据。传统的寻找逼近函数 F 的方法是通过以下最小化目标函数 (标准误差项) 实现的:

$$E_S(F) = \frac{1}{2} \sum_{i=1}^N (y_i - F(X_i))^2 \quad (13-7)$$

该函数体现了期望响应与实际响应之间的距离。而所谓的正则化方法, 是指在标准误差项基础上增加一个限制逼近函数复杂性的项 (正则化项), 该正则化项体现逼近函数的“几何”特性:

$$E_R(F) = \frac{1}{2} \|DF\|^2 \quad (13-8)$$

其中 D 为线性微分算子。于是正则化方法总的误差项定义为:

$$E(F) = E_S(F) + \lambda E_R(F) \quad (13-9)$$

其中, λ 为正则化系数 (正实数), 上述正则化问题的解为:

$$F(X) = \sum_{i=1}^N w_i G(X, X_i) \quad (13-10)$$

其中, $G(X, X_i)$ 为自伴随算子 $\tilde{D}D$ 的 Green 函数, w_i 为权系数。Green 函数 $G(X, X_i)$ 的形式依赖于算子 D 的形式, 如果 D 具有平移不变性和旋转不变性, 则 Green 函数值取决于 X 与 X_i 之间的距离, 即

$$G(X, X_i) = G(\|X - X_i\|) \quad (13-11)$$

如果选择不同的算子 D (应具有平移和旋转不变性), 便可得到不同的 Green 函数, 包括



Gaussian 函数这样最常用的径向基函数。

按照上述方式得到的网络称为正则化网络，其拓扑结构如图 13-5 所示。

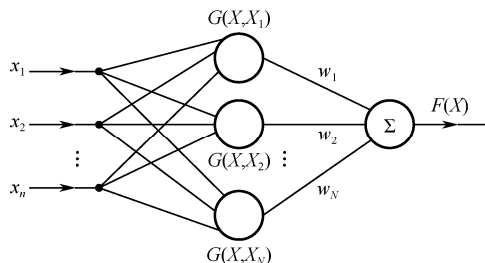


图 13-5 正则化网络

该正则化网络有以下特点：

- (1) 具有万能逼近能力，即只要有足够的隐节点，正则化网络能逼近紧集上的任意连续函数。
- (2) 具有最佳逼近特性，即任意给出未知的非线性函数 f ，总可以找到一组权系数，在该组系数下正则化网络对 f 的逼近优于其他系数。
- (3) 正则化网络得到的解是最优的，即通过最小化式 (13-9)，得到了同时满足对样本的逼近误差和逼近曲线平滑性的最优解。

13.3 RBF 神经网络的学习算法

RBF 神经网络的学习算法需要求解的参数有 3 个：基函数的中心、方差以及隐含层到输出层的权值。根据径向基函数中心选取方法的不同，RBF 神经网络有多种学习方法，如随机选取中心法、梯度训练法、有监督选取中心法和正交最小二乘法等。

13.3.1 自组织选取中心法

自组织选取中心法由两个阶段组成：一是自组织学习阶段，此阶段为无导师学习过程，求解隐含层基函数的中心与方差；二是有导师学习阶段，此阶段求解隐含到输出层之间的权值。

RBF 神经网络中常用的径向基函数为高斯函数，因此 RBF 神经网络的激活函数可表示为：

$$R(x_p - c_i) = \exp\left(-\frac{1}{2\sigma^2} \|x_p - c_i\|^2\right) \quad (13-12)$$

式中， $\|x_p - c_i\|$ 为欧氏范数， c_i 为高斯函数的中心， σ 为高斯函数的方差。

由图 13-4 所示的 RBF 神经网络的结构可得到网络的输出为：

$$y_j = \sum_{i=1}^h w_{ij} \exp\left(-\frac{1}{2\sigma^2} \|x_p - c_i\|^2\right) \quad (13-13)$$

式中， $x_p = (x_1^p, x_2^p, \dots, x_m^p)^T$ 为第 p 个输入样本； $p = 1, 2, \dots, P$ ， P 为样本总数； c_i 为网络



隐含层节点的中心； w_{ij} 为隐含层到输出层的连接权值； $i=1,2,\dots,h$ 为隐含层节点数； y_j 为与输入样本对应的网络的第 j 个输出节点的实际输出。

设 d 为样本的期望输出值，那么基函数的方差可表示为：

$$\sigma = \frac{1}{P} \sum_j^m \|d_j - y_j c_i\|^2 \quad (13-14)$$

学习算法具体步骤如下所述。

步骤1：基于K-均值聚类方法求取基函数中心 c 。

(1) 网络初始化：随机选取 h 个训练样本作为聚类中心 $c_i (i=1,2,\dots,h)$ 。

(2) 将输入的训练样本集合按最近邻规则分组：按照 x_p 与中心为 c_i 之间的欧氏距离将 x_p 分配到输入样本的各个聚类集合 $\mathcal{G}_p (p=1,2,\dots,P)$ 中。

(3) 重新调整聚类中心：计算各个聚类集合 \mathcal{G}_p 中训练样本的平均值，即新的聚类中心 c_i ，如果新的聚类中心不再发生变化，则所得到的 c_i 即RBF神经网络最终的基函数中心，否则返回(2)，进行下一轮的中心求解。

步骤2：求解方差 σ_i 。

该RBF神经网络的基函数为高斯函数，因此方差 σ_i 可由式(13-15)求解：

$$\sigma_i = \frac{c_{\max}}{\sqrt{2h}} \quad i=1,2,\dots,h \quad (13-15)$$

式中， c_{\max} 为所选取中心之间的最大距离。

步骤3：计算隐含层和输出层之间的权值。

隐含层至输出层之间神经元的连接权值可以用最小二乘法直接计算得到，计算公式如下：

$$w = \exp\left(\frac{h}{c_{\max}^2} \|x_p - c_i\|^2\right) \quad p=1,2,\dots,P; i=1,2,\dots,h \quad (13-16)$$

13.3.2 梯度训练方法

RBF神经网络的梯度训练方法[Hayk2001]与BP算法训练多层感知器的原理类似，也是通过最小化目标函数实现对各隐节点数据中心、扩展常数和输出权值的调节。这里给出一种带遗忘因子的单输出RBF神经网络学习方法，此时神经网络学习的目标函数为：

$$E = \frac{1}{2} \sum_{j=1}^N \beta_j e_j^2 \quad (13-17)$$

其中， β_j 为遗忘因子，误差信号 e_j 定义为：

$$e_j = y_j - F(X_j) = y_j - \sum_{i=1}^h w_i \Phi_i(X_j) \quad (13-18)$$

由于神经网络函数 $F(X)$ 对数据中心 c_i 、扩展常数 r_i 和输出权值 w_i 的梯度分别为：

$$\nabla_{c_i} F(x) = \frac{2w_i}{r_i^2} \Phi_i(X)(X - c_i) \quad (13-19)$$



$$\nabla_{r_i} F(x) = \frac{2w_i}{r_i^3} \Phi_i(X) \|X - c_i\|^2 \quad (13-20)$$

$$\nabla_{w_i} F(x) = \Phi_i(X) \quad (13-21)$$

考虑所有训练样本和遗忘因子的影响, c_i 、 r_i 和 w_i 的调节量为:

$$\Delta c_i = \eta \frac{w_i}{r_i^2} \sum_{j=1}^N \beta_j e_j \Phi_i(X_j) (X_j - c_i) \quad (13-22)$$

$$\Delta r_i = \eta \frac{w_i}{r_i^3} \sum_{j=1}^N \beta_j e_j \Phi_i(X_j) \|X_j - c_i\|^2 \quad (13-23)$$

$$\Delta w_i = \eta \sum_{j=1}^N \beta_j e_j \Phi_i(X_j) \quad (13-24)$$

其中, $\Phi_i(X_j)$ 为第 i 个隐节点对 X_j 的输出, η 为学习率。

13.3.3 正交最小二乘 (OLS) 学习算法

选择 RBF 神经网络数据中心的另一种方法是由 Chen 等人提出的正交最小二乘算法 (简称 OLS 算法)。该方法从样本输入中选取数据中心, 思路如下: 如果把所有样本输入均作为数据中心, 并令各扩展常数取相同值, 则根据前面提到的 Micchelli 定理, 隐层输出阵 $\mathbf{H} \in \mathbf{R}^{N \times N}$ 是可逆的, 于是目标输出 \mathbf{y} 可由 \mathbf{H} 的 N 个列向量线性表出。但是, \mathbf{H} 的 N 个列向量对 \mathbf{y} 的能量贡献显然是不同的, 因此我们可以从 \mathbf{H} 的 N 个列向量中按能量贡献大小依次找出 $M \leq N$ 个向量构成 $\hat{\mathbf{H}} \in \mathbf{R}^{N \times M}$, 直到满足给定误差 ε , 即

$$\|\mathbf{y} - \hat{\mathbf{H}}\mathbf{w}_0\| < \varepsilon \quad (13-25)$$

其中, \mathbf{w}_0 是使 $\|\mathbf{y} - \hat{\mathbf{H}}\mathbf{w}\|$ 最小的最优权向量 \mathbf{w} 的值。显然, 选择不同的 $\hat{\mathbf{H}}$, 式 (13-26) 的逼近误差是不同的。是否选择了一个最优的 $\hat{\mathbf{H}}$, 直接影响 RBF 神经网络的性能, 而一旦确定了 $\hat{\mathbf{H}}$, 也就确定了 RBF 神经网络的数据中心。

下面简要介绍能量贡献的计算原理。假定目标输出 \mathbf{y} 可由 N 个互相正交的向量 (不一定是单位向量) $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ 线性表出, 即

$$\mathbf{y} = \sum_{i=1}^N a_i \mathbf{x}_i \quad (13-26)$$

式 (13-26) 右乘 \mathbf{x}_i^T 后, 有

$$\mathbf{x}_i^T \mathbf{y} = a_i \|\mathbf{x}_i\|^2, i=1, 2, \dots, N \quad (13-27)$$

于是有,

$$\mathbf{y}^T \mathbf{y} = \sum_{i=1}^N a_i \|\mathbf{x}_i\|^2 \quad (13-28)$$

$$1 = \sum_{i=1}^N \frac{a_i \|\mathbf{x}_i\|^2}{\mathbf{y}^T \mathbf{y}} \quad (13-29)$$



因此, 选择 M 个基向量时的能量总贡献为:

$$g = \sum_{i=1}^M g_i = \sum_{i=1}^M \frac{a_i \| \mathbf{x}_i \|^2}{\mathbf{y}^T \mathbf{y}} \quad (13-30)$$

注意, 式(13-30)中 $0 \leq g_A \leq 1$ 。 M 越大, g_A 就越大, 逼近精度就越高。如果 $M=N$, 即选择了所有的基向量, 则逼近精度最高, 此时 $g_A=1$ 。

式(13-27)中的各 \mathbf{x}_i 是相互正交的, 而 \mathbf{H} 的各列并不正交, 因此正交最小二乘算法(OLS)对 \mathbf{H} 列的选择是在对 \mathbf{H} 作 Gram-Schmidt 正交化的过程中实现的。

Gram-Schmidt 正交化选择数据中心的步骤如下:

(1) 计算隐节点输出阵 \mathbf{H} , 并令 \mathbf{H} 的 N 个列向量为 $\mathbf{P}_1^1, \mathbf{P}_1^2, \dots, \mathbf{P}_1^N$, 它们构成 N 维欧氏空间 E_N^H 。

(2) 把输出数据向量 \mathbf{y} 投影到 $\mathbf{P}_1^1, \mathbf{P}_1^2, \dots, \mathbf{P}_1^N$ 上, 如果 \mathbf{y} 与某一个 \mathbf{P}_1^k 具有最大的夹角, 即 $\frac{\mathbf{y}^T \mathbf{P}_1^k}{\|\mathbf{y}\| \|\mathbf{P}_1^k\|}$ 的绝对值最大 (表示该 \mathbf{P}_1^k 对 \mathbf{y} 有最大能量贡献), 则把 \mathbf{P}_1^k 对应的样本输入选为第一个数据中心, \mathbf{P}_1^k 构成一维欧氏空间 E_1 。

(3) 用广义逆方法计算网络的输出权值 (包括偏移), 并得到网络对样本的训练误差。如果误差小于目标值则终止算法, 否则对前一步中剩下的 $N-1$ 个向量进行 Gram-Schmidt 正交化, 使之正交于 E_1 , 得到 $\mathbf{P}_2^1, \mathbf{P}_2^2, \dots, \mathbf{P}_2^{N-1}$ 。

(4) 找出与 \mathbf{y} 有最大投影的 \mathbf{P}_2^j , 选择与之对应的样本输入与第二个数据中心; 计算输出权值和训练误差, 并判断是否终止算法。

(5) 重复以上步骤, 直到 M 个数据中心使网络的训练误差小于给定值。

注意: 上述算法可以自动设计满足精度要求的网络结构。仿真发现, 尽管其结构不一定是最优的, 但网络规模确实是相对较小的。

13.4 其他 RBF 神经网络

13.4.1 广义回归神经网络

由于正则化网络的训练样本与基函数是一一对应的。当样本数 P 很大时, 实现网络的计算量将大得惊人。此外, P 很大则权值矩阵也很大, 求解网络的权值时容易产生病态问题。为了解决这一问题, 可减少隐节点的个数, 即 $N < M < P$, N 为样本维数, P 为样本个数, 从而得到广义 RBF 神经网络。

广义 RBF 神经网络的基本思想是: 用径向基函数作为隐单元的“基”, 构成隐含层空间。隐含层对输入向量进行变换, 将低维空间的模式变换到高维空间, 使得在低维空间内的线性不可分问题在高维空间内线性可分。

图 13-6 所示为 $N-M-l$ 结构的 RBF 神经网络, 即网络具有 N 个输入节点, M 个隐节点,

l 个输出节点, 且 $M < P$ 。 $X = (x_1, x_2, \dots, x_N)^T$ 为网络输入向量; $\varphi_j(X), (j=1, 2, \dots, M)$ 为任一隐节点的激活函数, 称为基函数, 一般选用格林 (Green) 函数; W 为输出权矩阵, 其中 $w_{jk}, (j=1, 2, \dots, M, k=1, 2, \dots, l)$, 为隐层第 j 个节点与输出层第 k 个节点间的突触权值; $T = (T_1, T_2, \dots, T_l)^T$ 为输出层阈值向量; $Y = (y_1, y_2, \dots, y_l)^T$ 为网络输出; 输出层神经元采用线性激活函数。

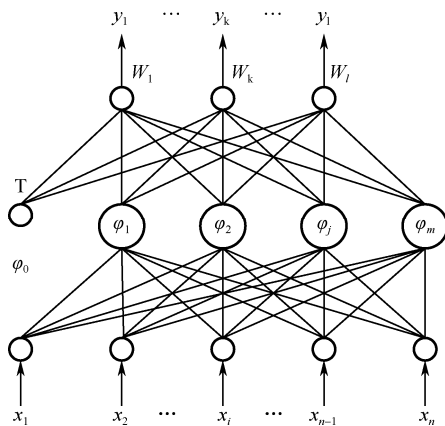


图 13-6 广义 RBF 神经网络

与正则化 RBF 神经网络相比, 广义 RBF 神经网络有以下几点不同:

- (1) 径向基函数的个数 M 与样本的个数 P 不相等, 且 M 常常远小于 P ;
- (2) 径向基函数的中心不再限制在数据点上, 而是由训练算法确定;
- (3) 各径向基函数的扩展常数不再统一, 其值由训练算法确定;
- (4) 输出函数的线性中包含阈值参数, 用于补偿基函数在样本集上的平均值与目标值平均值之间的差别。

13.4.2 泛化回归神经网络

泛化回归神经网络 GRNN (Generalized Regression NN) 常用于函数逼近, 其网络结构如图 13-7 所示, 它具有一个径向基网络层和一个特殊的线性网络层。

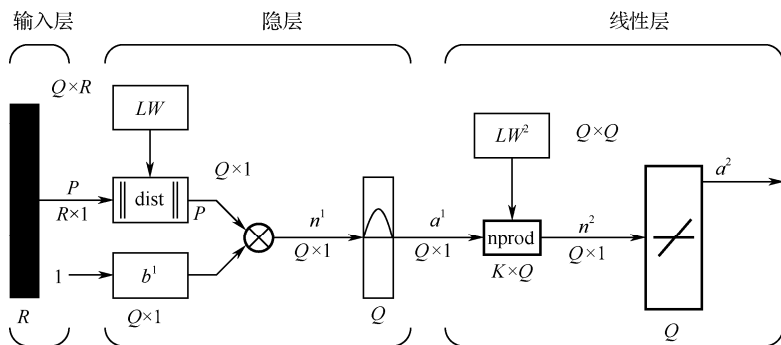


图 13-7 GRNN 模型



图中, 标有 nprod 的方框实现 LW^2 与 a^1 的归一化点乘运算 (以权值函数 normprod 完成), 其结果 n^2 为 LW^2 与 a^1 的点乘, 并以 a^1 所有元素的和进行归一化, 最后 n^2 作为线性神经元的加权输入。

GRNN 的第一层与 newrbe 创建的 RBF 一样, 其径向基神经元数目等于输入样本数, 其权值等于输入向量的转置; 阈值 $b = \frac{[-\log 0.5]^{\frac{1}{2}}}{\text{spread}}$ 。GRNN 第二层的神经元数也等于输入样本数, 其目标向量 T 无阈值向量, 并不需要训练。

13.4.3 概率神经网络

图 13-8 所示为概率神经网络 PNN 网络的结构图, 和径向基函数网络结构类似, 只是在第二层有细微差异。

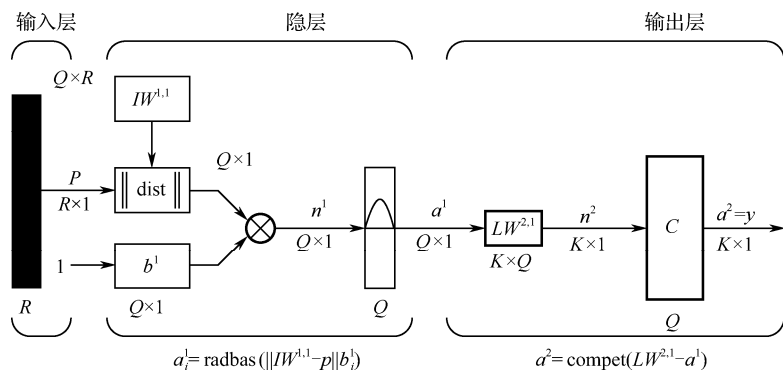


图 13-8 PNN 网络结构

对于图 13-8 图中 a_i^1 为向量 a^1 的第 i 个元素; $iIW^{1,1}$ 为权值矩阵 $IW^{1,1}$ 的第 i 行向量。

其中,

R 为输入向量元素的数目;

Q 为输入目标样本数目=第一层神经元的数目;

K 为输入向量类别数目=第二层神经元的数目。

如图 13-8 所示, PNN 网络第一层仍与前面两个网络类似, 首先计算输入向量与训练样本之间的距离, 第一层的输出向量表示输入向量与训练样本之间的接近程度。第二层将与输入向量相关的所有类别综合在一起, 网络输出为表示概率的向量, 最后通过第二层的竞争传递函数进行取舍, 概率最大值的那一类为 1, 其他类别用 0 表示。

假设输入期望值样本向量的数目为 Q , 期望值为 K 维向量, 表示类别只有一个元素为 1, 其余均为 0。

PNN 网络第一层的输入权值 $IW^{1,1}$ 为输入样本的转置矩阵 P' , 经过 $\|dist\|$ 计算, 第一层输出向量表示输入向量与训练样本向量接近的程度, 然后与阈值向量相乘, 再经过径向传递函数计算。输入向量与哪个输入样本最接近, 则神经元输出 a^1 对应元素为 1, 如果输入向量与几个类别的输入样本均接近, 则 a^1 相对应的几个元素均为 1。



第二层权值 $LW^{2,1}$ 设定为期望值向量矩阵 T ，每个行向量只有一个元素为 1，代表相应的类别，其余元素均为 0，然后计算乘积 Ta^1 。最后通过第二层传递函数竞争计算得到 n^2 ，较大的元素取值为 1，其余为 0。至此 PNN 网络就能够完成对输入向量的分类。

13.5 RBF 神经网络 MATLAB 函数

在 MATLAB 神经网络工具箱中提供了若干函数，用于实现 RBF 神经网络。下面分别给予介绍。

13.5.1 创建函数

在 MATLAB 神经网络工具箱中提供了 newrb、newrbe、newpnn 及 newgrnn 函数，用于创建 RBF 神经网络。

1) newrb 函数

该函数可以用来设计一个径向基网络，调用格式为：

```
net=newrb  
[net,tr]=newrb(P,T,GOAL,SPREAD,MN,DF)
```

其中，

P 为 Q 组输入向量组成的 $R \times Q$ 维矩阵；

T 为 Q 组目标分类向量组成的 $S \times Q$ 维矩阵；

GOAL 为径向基函数的扩展速度，默认为 1；

MN 为神经元的最大数目，默认为 Q ；

DF 为两次显示之间所添加的神经元数目，默认为 25；

net 为返回值，一个径向基网络；

tr: 返回值，训练记录。

注意：该函数设计的径向基网络 net 可用于函数逼近。径向基函数的扩展速度 SPREAD 越大，函数的拟合就越平滑。但是，过大的 SPREAD 意味着需要非常多的神经元以适应函数的快速变化。如果 SPREAD 设定得过小，则意味着需要许多神经元来适应函数的缓慢变化，这样一来，设计的网络性能就不会很好。因此，在网络设计过程中，需要用不同的 SPREAD 值进行尝试，以确定一个最优值。

【例 13-1】 利用 newrb 函数创建一个径向基网络。

```
>> P=[1 2 3];  
T=[2.0 4.1 5.9];  
net=newrb(P,T);  
P=1.5;  
Y=sim(net,P)
```



运行程序，输出如下，效果如图 13-9 所示。

```
NEWRB, neurons = 0, MSE = 2.54
NEWRB, neurons = 2, MSE = 0
NEWRB, neurons = 3, MSE = 0
Y =
    2.6755
```

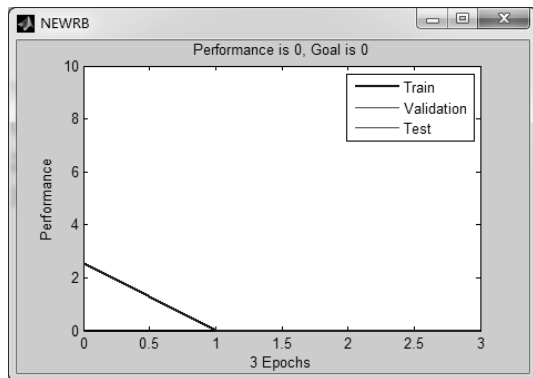


图 13-9 径向网络误差曲线

2) newrbe 函数

该函数用于设计一个准确的径向基网络，其调用格式为：

```
net=newrbe
net=newrbe(P,T,SPREAD)
```

各参数含义参见 newrb。

一般来讲，newrb 和 newrbe 一样，神经元数目越大，对函数的拟合就越平滑。但是，过多的神经元可能会导致计算困难。

注意：和 newrb 不同，newrbe 能够基于设计向量快速、无误差地设计一个径向基网络。

【例 13-2】 利用 newrbe 函数创建一个径向基函数，注意与例 13-1 的区别。

```
>> P = [1 2 3];
T = [2.0 4.1 5.9];
net = newrbe(P,T);
P = 1.5;
Y = sim(net,P)
```

运行程序，输出如下：

```
Y =
    2.8054
```

3) newpnn 函数

该函数可用于创建概率神经网络。概率神经网络是一种适用于分类问题的径向基网络，



其调用格式为:

```
net=newpnn  
net=newpnn(P,T,SPREAD)
```

各参数含义参见 newrb。

注意: 如果 SPREAD 值接近于 0, 则创建的概率神经网络可以作为一个最近邻域分类器。随着 SPREAD 值的增大, 需要更多考虑该网络附近的设计向量。

【例 13-3】 利用 newpnn 函数创建一个概率神经网络。

```
>> P=[1 2 3 4 5 6 7];  
Tc=[1 2 3 2 2 3 1];  
T=ind2vec(Tc)  
net=newpnn(P,T);  
Y=sim(net,P)  
Yc=vec2ind(Y)
```

运行程序, 输出如下:

```
T=  
    (1,1)      1  
    (2,2)      1  
    (3,3)      1  
    (2,4)      1  
    (2,5)      1  
    (3,6)      1  
    (1,7)      1  
  
Y=  
    1    0    0    0    0    0    1  
    0    1    0    1    1    0    0  
    0    0    1    0    0    1    0  
  
Yc=  
    1    2    3    2    2    3    1
```

4) newgrnn 函数

该函数可用于设计一个泛化回归神经网络。泛化回归神经网络是径向基网络的一种, 通常用于函数逼近, 其调用格式为:

```
net=newgrnn  
net=newgrnn(P,T,SPREAD)
```

各参数含义参见 newrb。

注意: 同 newrb 一样, SPREAD 的值越大, 由此设计的网络对函数的拟合就越平滑。为了更精确地对数据进行拟合, 最好使 SPREAD 的值小于输入向量之间的典型距离。

【例 13-4】 利用 newgrnn 函数创建一个泛化回归网络。

```
>> P=[1 2 3];
```



```
T = [2.0 4.1 5.9];  
net = newgrnn(P,T);  
P = 1.5;  
Y = sim(net,P)
```

运行程序，输出如下：

```
Y =  
    3.3667
```

13.5.2 权函数

在 MATLAB 神经网络工具箱中提供了 `dist`、`dotprod` 及 `normprod` 权函数。

1) `dist` 函数

该函数为欧几里得距离权函数，用于计算距离，其调用格式为：

```
Z = dist(W,P,FP)
```

其中，

W 为 $S \times R$ 权矩阵。

P 为 $R \times Q$ 的 Q 列输入矩阵。

FP 为函数参数的结构（可选的，忽略）。

Z 为返回的欧几里得距离。

【例 13-5】 计算随机矩阵的欧几里得距离。

```
>> W = rand(4,3);  
P = rand(3,1);  
Z = dist(W,P)
```

运行程序，输出如下：

```
Z =  
    0.2581  
    0.4244  
    1.0701  
    0.1863
```

2) `dotprod` 函数

该函数为点积权函数，其调用格式为：

```
Z = dotprod(W,P,FP)
```

其输入参数含义与 `dist` 函数类似，其中， Z 为返回的点积权。

【例 13-6】 用权函数 `dotprop` 计算随机矩阵的点积权。

```
>> W = rand(4,3);  
P = rand(3,1);
```



```
Z = dotprod(W,P)
```

运行程序，输出如下：

```
Z =  
    0.8445  
    0.7114  
    0.5123  
    0.9945
```

3) normprod 函数

该函数为规范点权函数，其调用格式为：

```
Z = normprod(W,P,FP)
```

其输入参数含义与 dist 函数类似，其中，Z 为返回的规范点积权。

【例 13-7】 用权函数 normprod 计算随机矩阵的规范点积权。

```
>> W = rand(4,3);  
P = rand(3,1);  
Z = normprod(W,P)
```

运行程序，输出如下：

```
Z =  
    0.4551  
    0.3923  
    0.4844  
    0.2053
```

13.5.3 输入函数

在 MATLAB 神经网络工具箱中提供了 netprod 函数，用于作为径向基网络的输入，其调用格式为：

```
N = netprod({Z1,Z2,...,Zn})
```

Netprod 函数是网络输入的积函数，网络输入函数结合它的加权输入和阈值计算一层的网络输出，其中 Z1,Z2,...,Zn 为 $S \times Q$ 矩阵。

【例 13-8】 netprod 函数的演示效果。

```
>> Z1 = [1 2 4; 3 4 1];  
Z2 = [-1 2 2; -5 -6 1];  
Z = {Z1,Z2};  
N = netprod({Z})  
B = [0; -1];  
Z = {Z1, Z2, concur(B,3)};  
N = netprod(Z)
```




运行程序，输出如下：

```
N =
    [2x3 double]    [2x3 double]
N =
     0     0     0
    15    24    -1
```

13.5.4 传递函数

在 MATLAB 神经网络工具箱提供了 `radbas` 函数，用于实现径向基网络的传递，其调用格式为：

```
A=radbas(N)
info=radbas(code)
```

其中，

N：输入（列）向量的 $S \times Q$ 维矩阵；

A：函数返回矩阵，与 **N** 一一对应，即 **N** 中的每个元素通过径向基函数得到 **A**；

info=radbas(code)：根据 **code** 值的不同返回有关函数的不同信息，包括：

deriv——返回导函数的名称；

name——返回函数全称；

output——返回输入范围；

active——返回可用输入范围。

13.5.5 mse 函数

该函数为均方误差性能函数，其调用格式为：

```
perf = mse(E,Y,X,FP)
```

其中，

E 为误差向量。

Y 为矩阵或细胞阵列输出向量（忽略）。

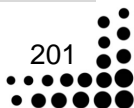
X 为所有权值和阈值组成的向量。

FP 为性能参数。

Perf 为返回的按照均方误差测量的网络性能值。

【例 13-9】 对所创建的径向基网络进行性能测试。

```
>> clear all;
P = [1 2 3];
T = [2.0 4.1 5.9];
net = newrbf(P,T);      %创建径向基网络
p = [-10 -5 0 5 10];
```





```

t = [0 0 1 1 1];
y = sim(net,p)      %网络仿真
e = t-y             %误差
perf = mse(e)        %均方误差性能

```

运行程序，输出如下：

```

y =
    1.4600    1.4600    1.5031    1.7211    1.4600
e =
   -1.4600   -1.4600   -0.5031   -0.7211   -0.4600
perf =
    1.0496

```

13.5.6 变换函数

变换函数包括 `ind2vec` 函数及 `ver2ind` 函数。

1) `ind2vec` 函数

该函数的功能用于将下标换成单值向量组，其调用格式为：

`ind2vec(ind)`

其中，`ind2vec` 函数输入为包含 n 个下标的行向量 `ind`，调用后可以得到 m 行 n 列的向量组矩阵，结果是矩阵中的每个向量 I ，除了 `ind` 中的第 I 个元素指定位置为 1 外，其余元素为 0，结果矩阵的行数等于 `ind` 中最大的下标值。

2) `ver2ind` 函数

该函数的功能是将单值向量变换成下标向量，其调用格式为：

`ver2ind(vec)`

`ver2ind` 和 `ind2vec` 函数互为逆变化，`ver2ind` 函数输入为一个 m 行 n 列的向量矩阵 `vec`，调用后可以得到 n 个下标值大于等于 0 的行向量，`x` 中的每个向量 I 除包含一个 1 外，其余均为 0，得到的行向量包含这些非 0 元素的下标。

【例 13-10】 变换函数用法演示。

```

>> clear all;
P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3];
Tc = [1 1 2 2 3 3 3];
T = ind2vec(Tc)           %将下标换为向量形式
net = newpnn(P,T);
Y = sim(net,P);
Yc = vec2ind(Y)

```

运行程序，输出如下：



T =

(1,1)	1
(1,2)	1
(2,3)	1
(2,4)	1
(3,5)	1
(3,6)	1
(3,7)	1

Yc =

1	1	2	2	3	3	3
---	---	---	---	---	---	---

第 14 章 Simulink 神经网络应用

Simulink 工具箱包含大量的动态仿真库，能够对实际系统进行动态仿真，而且可以非常方便地实现 Simulink 与 MATLAB 之间的交互操作，使 MATLAB 命令进行 Simulink 模型的仿真、数据交换等，同时可以编写 M 函数或者 S 函数进行复杂系统的 Simulink 仿真。在此主要介绍 Simulink 建模与仿真的基本知识及应用实例分析。

14.1 Simulink 神经网络仿真模型库

在 MATLAB 命令窗口中输入 `neural` 即可打开神经网络仿真模型库 (Library:neural) 窗口，如图 14-1 所示。

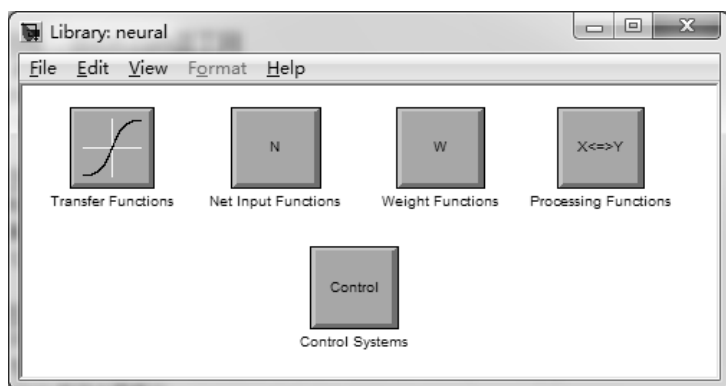


图 14-1 神经网络仿真模型库

从图 14-1 中可看出，神经网络工具箱包括 5 个模块库，分别为 Transfer Functions（传递函数模块库）、Net Input Functions（网络输入模块库）、Weight Functions（权值设置模块库）、Processing Functions（处理模块库）及 Control Systems（控制系统模块库）。下面分别对这 5 类模块库进行介绍。

1. Transfer Functions 模块

在图 14-1 中，双击 Transfer Functions 模块，即可弹出 Library:neural/Transfer Functions 窗口，如图 14-2 所示。

其中包括的传递函数模块如表 14-1 所列。

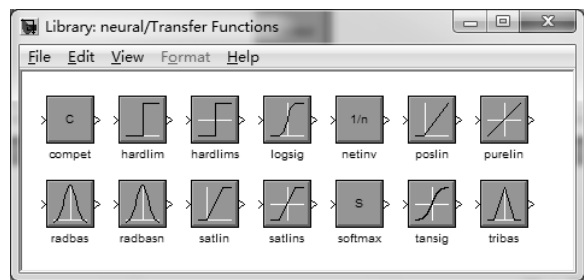


图 14-2 Transfer Functions 模块

表 14-1 Transfer Functions 子模块库包括的传递函数模块

模 块	功 能 描 述
compet	竞争传递函数
hardlim	强限幅传递函数
hardlims	对称强限幅传递函数
logsig	logsig 传递函数 (log-S 型传递函数)
netinv	净逆传递函数
poslin	正线性传递函数
purelin	纯线性传递函数
radbas	径向基传递函数
radbasn	归一化径向基传递函数
satlin	饱和和线性传递函数
satlins	对称饱和和线性传递函数
softmax	弱最大值传递函数
tansig	tansig 传递函数 (双曲正切 S 型传递函数)
tribas	三角基传递函数

传递函数模块库的每个模块都是一种激励函数，能够接收一个网络输入向量，并且产生一个相应的输出向量。

2. Net Input Function 模块

双击图 14-1 中的 Net Input Function 模块，即可弹出 Library:neural/Net Input Functions 窗口，如图 14-3 所示。

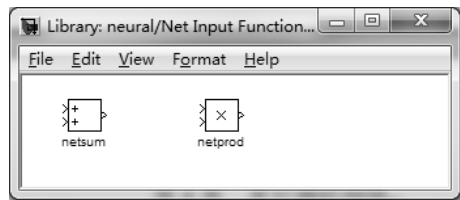


图 14-3 Net Input Function 模块

其中包括的网络输入模块如表 14-2 所列。

表 14-2 Net Input Function 模块库包括的网络输入模块

模 块	功 能 描 述	模 块	功 能 描 述
netsum	相加或相减计算	netprod	点乘或点除计算

每一个模块都能够接收任意数目的加权输入向量、加权的层输出向量或者阈值向量，并且返回一个网络输入向量。

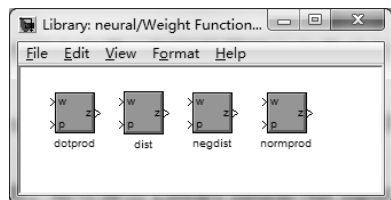


图 14-4 Weight Functions 模块

这两个模块与 Simulink 中 Math operations 模块库中的 sum 和 product 模块功能相似。

3. Weight Functions 模块

双击图 14-1 中的 Weight Functions 模块，即可弹出 Library:neural/Weight Functions 窗口，如图 14-4 所示。

其中包括的网络输入模块如表 14-3 所列。

表 14-3 Weight Functions 模块库包括的网络输入模块

模 块	功 能 描 述	模 块	功 能 描 述
dotprod	点乘权值函数	negdist	Euclidean 距离的负值计算权值函数
dist	距离权值函数	normprod	规范化的点乘权值函数

权值模块库中的每个模块都以一个神经元权值向量作为输入，并将其与一个输入向量（或者是某一层的输出向量）进行计算，得到神经元的加权输入值

4. Processng Functions 模块

双击图 14-1 中的 Processng Functions 模块，即可弹出 Library:neural/Processng Functions 窗口，如图 14-5 所示。

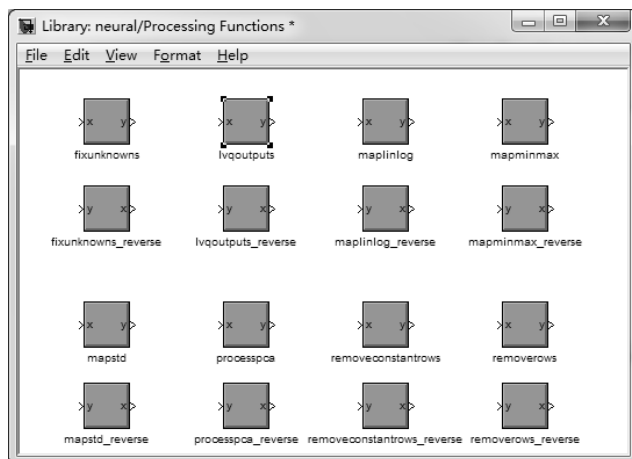


图 14-5 Processng Functions 模块

其中包括的处理模块如表 14-4 所列。

表 14-4 Processing Functions 模块库包括的处理模块

模 块	功 能 描 述
fixunknowns	通过未知的标识行处理数据过程
lvqoutputs	基于 LVQ 输出预处理
maplinlog	通过映射矩阵 X 中每个元素的线性值到 log 值
mapminmax	通过映射矩阵行的最低及最高值[-1, 1]处理矩阵
mapstd	通过映射矩阵每一行的平均值 0 及方差 1 处理矩阵
processpca	使用主成分分析法处理矩阵的列
removeconstantrows	通过删除矩阵的列及矩阵常量处理矩阵
removerows	通过删除矩阵的列及矩阵指定索引处理矩阵
lvqoutputs_reverse	反向基于 LVQ 输出预处理
fixunknowns_reverse	反向通过未知的标识行处理数据过程
mapminmax_reverse	反向通过映射矩阵行的最低及最高值[-1, 1]处理矩阵
Maplinlog_reverse	反射通过映射矩阵 X 中每个元素的线性值到 log 值
mapstd_reverse	反向通过映射矩阵每一行的平均值 0 及方差 1 处理矩阵
processpca_reverse	反向使用主成分分析法处理矩阵的列
removeconstantrows_reverse	反向通过删除矩阵的列及矩阵常量处理矩阵
removerows_reverse	反向通过删除矩阵的列及矩阵指定索引处理矩阵

5. Control Systems 模块

双击图 14-1 中的 Control Systems 模块，即可弹出 Library:neural/Control Systems 窗口，如图 14-6 所示。

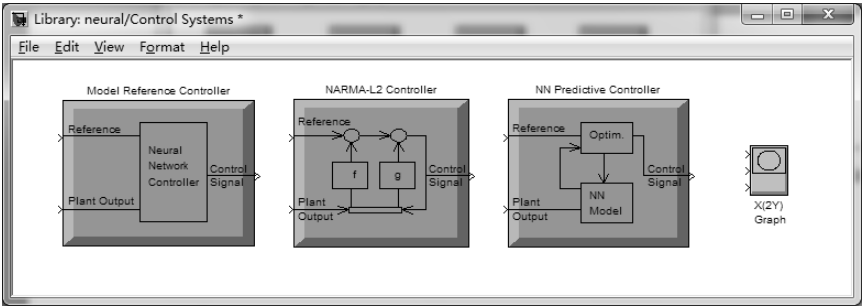


图 14-6 Control Systems 模块

其中包括的控制系统模块如表 14-5 所列。

表 14-5 Control Systems 包括的控制系统模块

模 块	功 能 描 述	模 块	功 能 描 述
Model Reference Controller	模块参考控制器	NN Predictive Controller	神经网络预测控制器
NARMA-L2 Controller	NARMA-L2 控制器	Graph	示波器



控制系统有 4 个模块，前 3 个模块为控制器，最后一个模块为示波器。

14.2 Simulink 神经网络应用

在 MATLAB 中提供了 `gensim` 函数，用于对一个网络生成的模块化进行描述，使得读者能够方便地在 Simulink 中对模型进行设计和对网络进行仿真，其调用格式为：

`gensim(net,st)`：其中 `net` 为神经网络；`st` 为样本时间，默认值为 1。如果 `net` 没有输入或相关层的延迟，即 `net.numInputDelays` 和 `net.numLayerDelays` 均为 0，即可使用设定 `st=-1` 来得到一个连续取样网络。

【例 14-4】 通过测试 $y=3x$ 来说明 `gensim` 函数用法。

```
>> clear all;
%定义一个输入 P 和相应的目标 T
x=[1 2 3 4 5 6];
y=3*x;
%利用 newlind 函数设计一个线性神经网络
net=newlind(x,y)
net =
    Neural Network                                %神经网络
        name: 'Linear Designed'
        efficiency: .cacheDelayedInputs, .flattenTime,
                  .memoryReduction
        userdata: (your custom info)

        dimensions:                                %尺寸
            numInputs: 1
            numLayers: 1
            numOutputs: 1
            numInputDelays: 0
            numLayerDelays: 0
            numFeedbackDelays: 0
            numWeightElements: 2
            sampleTime: 1

        connections:                                %连接
            biasConnect: true
            inputConnect: true
            layerConnect: false
            outputConnect: true

        subobjects:                                %子对象
```




```

        inputs: { 1x1 cell array of 1 input}
        layers: { 1x1 cell array of 1 layer}
        outputs: { 1x1 cell array of 1 output}
        biases: { 1x1 cell array of 1 bias}
        inputWeights: { 1x1 cell array of 1 weight}
        layerWeights: { 1x1 cell array of 0 weights}

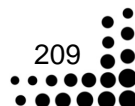
functions:                                     %函数
    adaptFcn: (none)
    adaptParam: (none)
    derivFcn: 'defaultderiv'
    divideFcn: (none)
    divideParam: (none)
    divideMode: 'sample'
    initFcn: 'initlay'
    performFcn: (none)
    performParam: (none)
    plotFcns: {}
    plotParams: { 1x0 cell array of 0 params}
    trainFcn: (none)
    trainParam: (none)

weight and bias values:                       %权值与阈值
    IW: { 1x1 cell} containing 1 input weight matrix
    LW: { 1x1 cell} containing 0 layer weight matrices
    b: { 1x1 cell} containing 1 bias vector

methods:                                       %方法
    adapt: Learn while in continuous use
    configure: Configure inputs & outputs
    gensim: Generate Simulink model
    init: Initialize weights & biases
    perform: Calculate performance
    sim: Evaluate network outputs given inputs
    train: Train network with examples
    view: View diagram
    unconfigure: Unconfigure inputs & outputs

evaluate:                                     %评估
    outputs = net(inputs)
>> %输入测试数据
test=[1.5 2.5 3.5 4.5 5.5 6.5];

```



```
%使用 sim 函数测试网络
y=sim(net,test)
y =
    4.5000    7.5000   10.5000   13.5000   16.5000   19.5000
>> %调用 gensim 函数创建 net 网络模型
gensim(net,-1)           %其中-1 表示将生成一个连续采样网络模块
```

运行程序，生成如图 14-7 所示的 Simulink 模型。模型中包括一个线性网络的仿真，这个线性网络输入端连接一个采样输入，输出端连接一个示波器。

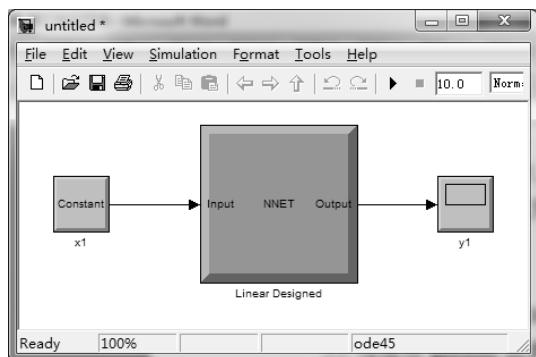


图 14-7 系统模型创建窗口

由图 14-7 可看出，在该窗口中，已经建立了神经网络模块（Linear Designed），另外，还有一个采样输入 x1 和一个示波器输出 y1。

选择图 14-7 中的 Linear Designed 模块并右击，在弹出的快捷菜单中选择“look Under Mask”项，即可弹出如图 14-8 所示的窗口。

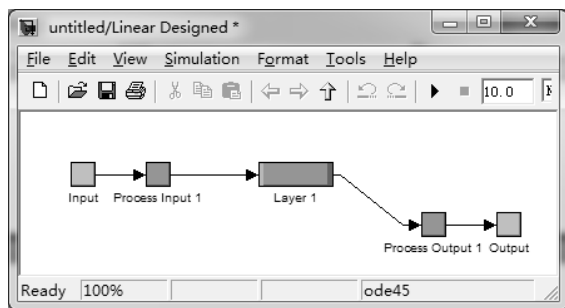


图 14-8 Linear Designed 模块网络结构窗口

如果用户还想了解更详细的网络结构，可在弹出的窗口中双击需要了解的模块，如图 14-9 所示的 Layer1 模块结构。这样一直下去，直到出现的窗口为属性设置窗口为止。

如果不做任何修改，对图 14-7 进行仿真，选择【Simulation】菜单下的【Start】选项进行仿真，得到输出波形如图 14-10 所示。

由图 14-10 可看出，该波形并不能反映峰值检波的过程，这是因为输入向量只有一个常量。如果要观察动态检波过程，则需要对系统模型进行修改。

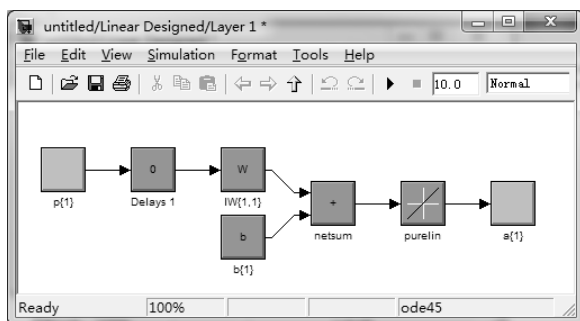


图 14-9 Layer1 模块结构图

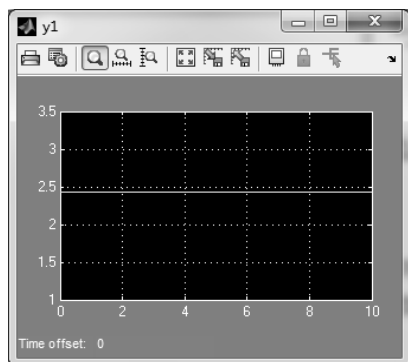


图 14-10 原系统仿真结果图

首先在 MATLAB 命令窗口中输入 `simulink`，打开 Simulink Library Browser 浏览器，在 Source 模块库中把 Signal Generator 模块拖放到模型窗口中，用于替换原输入模块，其效果如图 14-11 所示。

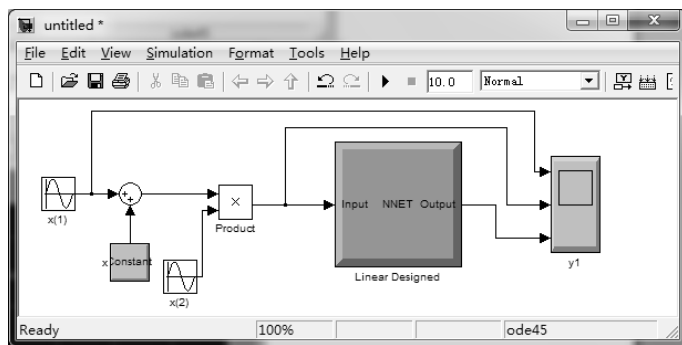


图 14-11 神经网络峰值检波动态仿真模型

图中，信号源 $x(1)$ 为调制信号，频率为 1rad/s ；信号源 $c(1)$ 为载波信号，频率为 15rad/s ； $AM(1)$ 为已调波信号； $y(1)$ 为峰值检波的输出信号。示波器绘出了 $s(1)$ ， $AM(1)$ 和 $y(1)$ 的波形 (Wave)，如图 14-12 所示。

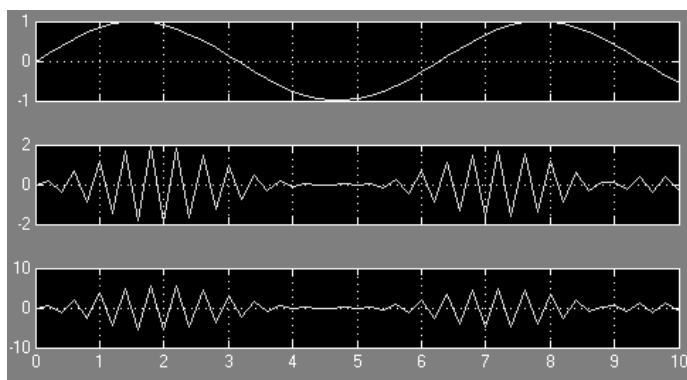


图 14-12 动态仿真效果图



从以上过程可看出，在 Simulink 环境中对神经网络进行动态仿真的一般步骤为：

- 在命令窗口或以程序完成神经网络的设计与训练；
- 以 gensim(net,st)函数生成包括神经网络模块在内的动态仿真模型；
- 以 Simulink 命令打开 Simulink Library Browser 窗口，根据仿真欲达到的目的，按照 Simulink 的一般操作方法，修改系统模型。
- 以 Simulink 进行仿真，输出动态仿真结果。

第 15 章 ART 网络与 CP 网络算法分析与应用

随着计算机技术的飞速发展，通过建立生理系统仿真模型再现真实系统的状态和运行过程，为深度探究系统规律开拓了一条高效率的崭新之路，作为一种系统建模的方法与工具，已经在自然科学和工程技术领域获得了广泛的应用。仿真模型在生理系统领域的崭露头角是该项技术的跨领域应用和扩展。自适应共振理论 ART（Adaptive Resonance Theory）模型是美国 Boston 大学的 S. Grossberg 和 A. Carpenet 在 1976 年提出的。ART 是一种自组织神经网络结构，是无教师的学习网络。当神经网络和环境有交互作用时，对环境信息的编码会自发地在神经网中产生，则认为神经网络在进行自组织活动。ART 就是这样一种能自组织地产生对环境认识编码的神经网络理论模型。

如何保证在适当增加网络规模的同时，在过去记忆的模式和新输入的训练模式之间作出某种折中，既能最大限度地接收新的模式信息（灵活性），同时又能保证较少地影响过去的模式样本（稳定性）呢？ART 网较好地解决了稳定性和灵活性兼顾的问题。

ART 网络共有 3 种类型，ART-1、ART-2 和 ART-3。这里主要介绍第一类型网络。

15.1 ART-1 型网络

15.1.1 ART-1 型网络结构

从图 15-1 给出的 ART 模型结构可以看出，该模型的结构与前面出现过的网络拓扑结构有较大区别。ART-1 网络由两层神经元构成两个子系统，分别称为比较层 C（或称注意子系统）和识别层 R（或称取向子系统）。此外还有 3 种控制信号：复位信号（Reset），逻辑控制信号 G_1 和 G_2 。

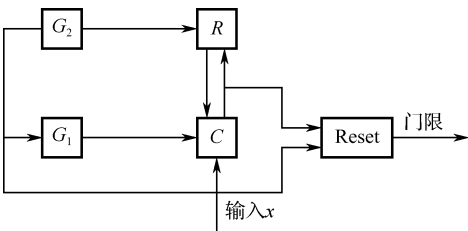


图 15-1 RT I 型网络结构

下面对图 15-1 中各部分功能进行介绍。

1) C 层结构

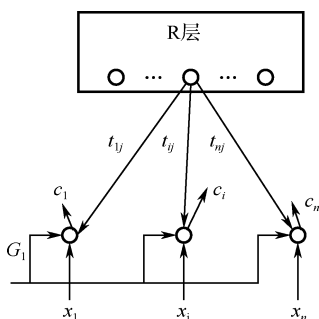


图 15-2 比较层结构示意图

C 层展开后的结构如图 15-2 所示，该层有 n 个神经元，每个神经元接收来自 3 个方面的信号：一个是来自外界的输入信号 x ；另一个是来自 R 层获胜神经元的外星权向量 T_{j^*} 的返回信号 t_{ij^*} ；还有一个是来自 G_1 的控制信号。 C 层神经元的输出是根据 2/3 的“多数表决”原则产生的，即输出值 c_i 与 x_i 、 t_{ij^*} 、 G_1 3 个信号中的多数信号值相同。

网络开始运行时， $G_1=1$ ，识别层尚未产生竞争获胜神经元，因此反馈回送信号为 0。由 2/3 规则知， C 层输出应由输入信号决定，有 $C=X$ 。当网络识别层出现反馈回送信号时， $G_1=0$ ，由 2/3 规则知， C 层输出应取决于输入信号与反馈信号的比较情况，如果 $x_i = t_{ij^*}$ ，则 $c_i = x_i$ ，否则 $c_i = 0$ 。可以看出，控制信号 G_1 的作用是使比较层能够区分网络运行的不同阶段，网络开始运行阶段 G_1 的作用是使 C 层对输入信号直接输出；之后 G_1 的作用是使 C 层行使比较功能，此时 c_i 为对 x_i 和 t_{ij^*} 比较，两者同时为 1 时， c_i 为 1，否则为 0，可以看出，从 R 层返回的信号 t_{ij^*} 对 C 层输出有调节作用。

2) R 层结构

R 层展开后的结构如图 15-3 所示，其功能相当于一种前馈竞争网。设 R 层有 m 个神经元，用于表示 m 个输入模式类。 m 可动态增长，以设立新模式类。由 C 层向上连接到 R 层第 j 个神经元的内星权向量用 $B_j = (b_{1j}, b_{2j}, \dots, b_{nj})$ 表示。 C 层的输出向量 C 沿 m 个内星权向量 $B_j (j=1, 2, \dots, m)$ 向前传送，到达 R 层各个神经元后经过竞争再产生获胜神经元 j^* ，指示本次输入模式的所属类别。获胜神经元输出 $r_{j^*}=1$ ，其余神经元输出为 0。 R 层的每个神经元都对应两个权向量，一个是将 C 层前馈信号汇聚到 R 层的内星权向量 B_j ，另一个是将 R 层反馈信号散发到 C 层的外星权向量 T_j ，该向量为对应于 R 层各神经元的存储模式类。

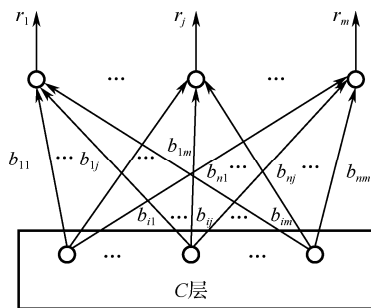


图 15-3 识别层结构示意图

3) 控制信号

3 个控制信号的作用分别是：信号 G_2 检测输入模式 X 是否为 0，它等于 X 各分量的逻辑“或”，如果 $x_i (i=1, 2, \dots, n)$ 全为 0，则 $G_2=0$ ，否则 $G_2=1$ 。设 R 层输出向量各分量的逻辑“或”用 R_0 表示，则信号 $G_1 = G_2 R_0$ ，当 R 层输出向量各分量全为 0 而输入向量不是零向量时， $G_1=1$ ，否则 $G_1=0$ 。正如前面所指出的， G_1 的作用是在网络开始运行时为 1，以使 $C=X$ ，其后为 0，以使 C 值由输入模式和反馈的存储模式的比较结果决定。Reset 信号的作用是使 R 层竞争获胜



神经元无效, 如果根据某种事先设定的测量标准, T_{j^*} 与 X 未达到预先设定的相似度 ρ , 表明两者未充分接近, 于是系统发出 Reset 信号使竞争获胜神经元无效。

15.1.2 ART-1 网络学习过程

ART-1 网络的学习过程可以归纳如下。

(1) 初始化。令 $t_{ij}(0)=1$, $w_{ij}(0)=\frac{1}{N+1}$, $i=1,2,\dots,N$, $j=1,2,\dots,M$ 。其中, 警戒参数 $0 < \rho \leq 1$ 。

(2) 将输入模式 $A_k = (a_1^k, a_2^k, \dots, a_N^k)$ 提供给网络的输入层。

(3) 计算输出层各个神经元的输入加权和。

$$s_j = \sum_{i=1}^N w_{ij} a_i^k, \quad j=1,2,\dots,M$$

(4) 选择输入模式的最佳分类结果:

$$s_g = \max_{j=1,2,\dots,M} s_j$$

令神经元 g 的输出为 t 。

(5) 计算以下 3 式, 并进行判断:

$$\begin{aligned} |A_k| &= \sum_{i=1}^N a_i^k \\ |T_g A_k| &= \sum_{i=1}^N t_{gi} a_i^k \\ \frac{|T_g A_k|}{|A_k|} &> \rho \end{aligned}$$

如果最后一式成立, 则转入步骤 (7), 否则转入步骤 (6)。

(6) 取消识别结果, 将输出层神经元 g 的输出值复位为 0, 并将这一神经元排除在下次识别的范围之外, 返回步骤 (4)。当所有已利用过的神经元都无法满足步骤 (5) 中的最后一式时, 则选择一个新的神经元作为分类结果, 并进入步骤 (7)。

(7) 接受识别结果, 并调整连接权值:

$$\begin{aligned} w_{ig}(t+1) &= \frac{t_{gi}(t) a_i}{0.5 + \sum_{i=1}^N t_{gi}(t) a_i} \\ t_{gi}(t+1) &= t_{gi}(t) a_i \end{aligned}$$

其中, $i=1,2,\dots,N$ 。

(8) 将步骤 (6) 中复位的所有神经元重新加入识别范围中, 返回步骤 (2) 对下一个模式进行识别。

无论网络学习还是回想, 都使用以上的规则。只不过在网络回想时, 只对那些与未使用过的输出神经元有关的连接权值向量 w_{ij} 和 t_{ij} 才进行初始化。其他连接权值向量保持网络学习后的值不变。当输入模式为一个网络已记忆的学习模式时, 不需要再进行网络权值的调整,



这是因为当输入模式和网络记忆的学习模式完全相等，再按照权值调整公式进行调整时，网络连接权值不会发生任何变化。而当输入模式与网络记忆模式存在一定差异时，按照权值调整公式进行调整，将会影响网络原有模式的记忆效果。但是如果输入的是一个全新的模式，需要利用网络对其另外记忆时，则必须按照权值调整公式对网络连接权值进行调整。

尽管 ART-1 网络具有许多其他网络所没有的优点，但是它仅以输出层中某个神经元代表分类结果，而不是像 Hopfield 网络那样，把分类结果分散在各个神经元上来表示。所以，一旦输出层中某个输出神经元损坏，则会导致该神经元所代表类别的模式信息全部消失。这是 ART-1 网络一个很大的缺陷。

15.1.3 ART-1 网络的应用

MATLAB 神经网络工具箱没有为 ART 型网络提供专门的函数，因此，利用现有的神经网络工具箱是无法实现 ART-1 网络的。但是，我们可以借助于 MATLAB 强大的数学计算功能来实现 ART-1 网络的训练和联想记忆功能。

【例 15-1】 现举一个简单的例子来演示利用 MATLAB 实现 ART-1 网络的过程。如图 15-4 所示，设 ART-1 网络有 5 个输入神经元和 20 个输出神经元，现有两组输入模式 $A_1 = (1, 1, 0, 0, 0)$ 和 $A_2 = (1, 0, 0, 0, 1)$ ，要求利用这两组模式来训练网络。根据 15.1.2 节中的训练过程，该网络的训练步骤为下面几步。

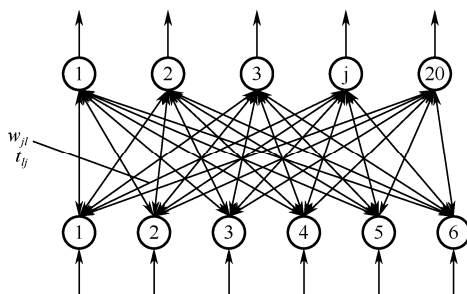


图 15-4 ART-1 网络实例

- (1) 初始化。令 $w_{ij} = 1/n + 1 = 1/6$ ， $t_{ji} = 1$ ，其中 $i = 1, 2, \dots, 5$ ， $j = 1, 2, \dots, 20$ ，令 $\rho = 0.8$ 。
- (2) 将输入模式 A_1 提供给网络的输入层。
- (3) 求获胜的神经元。因为在网络的初始状态下，所有的前馈连接权 w_{ij} 均取相等的权值 $1/6$ ，所以各输入神经元均具有相同的输入加权和 s_j 。这时可取任一个神经元作为 A_1 的分类代表，如第 1 个，令其输出值为 1。

$$(4) \text{ 计算 } |A_1| = \sum_{i=1}^5 a_i = 2, \quad |T_1 A_1| = \sum_{i=1}^5 t_{1i} a_i = 2。$$

$$(5) \text{ 计算 } \frac{|T_1 A_1|}{|A_1|} = 1 > 0.8, \text{ 接受这次识别结果。}$$

- (6) 调整权值。

$$W_1 = (w_{11}, w_{12}, w_{13}, w_{14}, w_{15}) = (0.4, 0.4, 0, 0, 0)$$



$$T_1 = (t_{11}, t_{21}, t_{31}, t_{41}, t_{51}) = (1, 1, 0, 0, 0)$$

至此， A_1 已经被记忆在网络中了。

(7) 将输入模式 A_2 提供给网络的输入层。

(8) 求获胜神经元， $s_1 = 0.4$ ， $s_2 = 1/6$ ， $s_3 = \dots = s_{20} = 1/6$ ，由于 $s_1 > s_2 = s_3 = \dots = s_{20}$ ，所以取神经元 1 作为获胜神经元，但这显然与 A_1 的识别结果相矛盾。又因为：

$$\frac{|T_2 A_2|}{|A_2|} = \frac{1}{2} < 0.8$$

所以拒绝这次识别结果，重新进行识别。由于 $s_2 = s_3 = \dots = s_{20} = 1/6$ ，故可从中任选一个神经元作为 A_2 的分类结果，如神经元 20。

(9) 调整权值。

$$W_2 = (w_{21}, w_{22}, w_{23}, w_{24}, w_{25}) = (0.4, 0, 0, 0, 0.4)$$

$$T_2 = (t_{12}, t_{22}, t_{32}, t_{42}, t_{52}) = (1, 0, 0, 0, 1)$$

至此， A_2 也记忆在网络中了。

按照上述步骤，可以编写以下 MATLAB 代码。

```
%竞争层的输出
xiu=rands(20);
%正向权值 W 和反向权值 T
W=rands(20,5);
T=rands(20,5);
%警戒参数
xiuxiu=0.8;
%两组模式 A1 和 A2
A1=[1 1 0 0 0];
A2=[1 0 0 0 1];
%初始化
for i=1:20
    for j=1:5
        W(i,j)=1/6;
        T(i,j)=1;
    end
end
%判定是否接受识别结果
normalA1=norm(A1,1);
normalTA1=T(1,:)*A1';
count=1;
if normalTA1/normalA1>xiuxiu
    xiu(count)=1;
end
%权值调整
W(1,:)=0.4 0.4 0 0 0;
```



```

T(1,:)=[1 1 0 0 0];
% 寻找可以记忆 A2 的神经元
for k=1:20
    s(k)=W(k,:)*A2';
    if s(k)==max(s)
        count=k;
    end
end
% 如果和 A1 的神经元重复, 继续寻找
if xiu(count)==1
    newcount=count+1
end
for i=1:(count-1)
    p(i)=s(i);
end
for i=count:19
    p(i)=s(i+1);
end
for k=newcount:20
    if s(k)==max(p)
        count=k;
    end
end
% 确定找到的神经元序号 count, 并令其对应的输出为 1
xiu(count)=1;
% 权值调整
W(count,:)= [0.4,0,0,0,0.4];
T(count,:)= [1,0,0,0,1];
xiu'

```

运行结果为:

```

xiu' =
    1.0000    0.6375   -0.1397    0.7806    0.4698    0.3746   -0.3078   -0.6679   -0.6888
-0.6178  -0.1551    0.7120   -0.0195    0.6319   -0.0785   -0.0853   -0.0986   -0.1756    0.8032
1.0000

```

可见, 第 1 个和第 20 个神经元的输出均为 1, 表示它们记忆了输入模式。

15.2 ART-2 型网络

ART-2 型网络与 ART-1 型的主要区别是 ART-2 型网络以模拟量作为输入模式, 同时在算法上做了一些相应的改进, 并采用慢速学习方式, 其抗干扰能力大大增强。ART-3 型网络是



由多个 ART-1 型网络组成的复合阶层型网络。

ART-2 神经网络不仅能对双极性或二进制输入模式分类, 而且能够对模拟输入模式的任意序列进行自组织分类, 其基本设计思路仍然是采用竞争学习策略和自稳机制。

15.2.1 网络结构与运行原理

ART-2 神经网络结构如图 15-5 所示, 图 15-6 中给出第 i 个处理单元的拓扑连接。ART-2 由注意子系统和取向子系统组成。注意子系统中包括短期记忆 ATM 特征表示场 F_1 和短期记忆类别表示场 F_2 。 F_1 相当于 ART-1 中的比较层, 包括几个处理级和增益控制系统。 F_2 相当于 ART-1 中的识别层, 负责对当前输入模式进行竞争匹配。 F_1 和 F_2 共有 N 个神经元, 其中 F_1 场有 M 个, F_2 场有 $N-M$ 个, 共同构成了 N 维状态向量代表网络的短期记忆。 F_1 和 F_2 之间的内外星连接向量构成了网络的自适应长期记忆 LTM, 由下至上的权值用 z_{ij} 表示, 由上至下的权值用 z_{ji} 表示。取向子系统由图 15-6 左侧的复位系统组成。

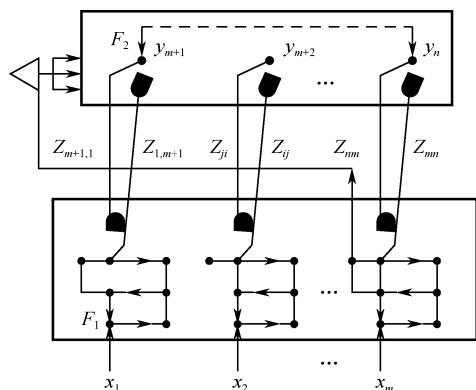


图 15-5 ART-2 神经网络

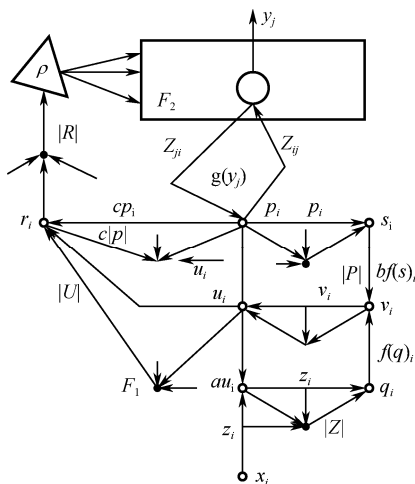


图 15-6 ART-2 网络拓扑示意图

F_1 场的 M 个神经元从外界接受输入模式 X , 经场内的特征增强与噪声抑制等处理后通过由下至上的权值 z_{ij} 送到 F_2 场。 F_2 场的 $N-M$ 个神经元接收 F_1 场上传的信号, 经过竞争确定哪个神经元获胜, 获胜神经元被激活, 其他则均被抑制。与激活神经元相连的内外星权向量进行调整。增益控制系统负责比较输入模式与 F_2 场激活神经元的外星权向量之间的相似程度, 当两向量的相似度低于警戒门限时, 复位子系统发出信号抑制 F_2 场的激活神经元。网络将在 F_2 场另选一个获胜神经元, 直到相似度满足要求。如果 F_2 场的神经元数 $N-M$ 大于可能的输入模式类别数, 总可以为所有新增的模式类分配一个代表神经元。

结合图 15-5 和以上分析看出, ART-2 网中有两种存储机制、两种连接权和两种抑制信号。两种存储机制是指: 长期记忆—— F_1 与 F_2 之间的权值; 短期记忆—— F_1 与 F_2 中的神经元状态。两种连接权是指: $F_1 \rightarrow F_2$ 的内星权, 决定 F_2 中哪个神经元获胜; $F_1 \rightarrow F_2$ 的外星权, 用作 F_1 输入模式的类别编码。两种抑制信号是指: F_1 场神经元的抑制信号, 来自增益控制子系统。



统； F_2 场神经元的抑制信号，来自复位子系统。

从以上分析可知，ART-2 与 ART-1 的原理类似，主要区别是 ART-2 的比较层 F_1 场的结构与功能更为复杂一些。

15.2.2 网络的数学模型与学习算法

1) 特征表示场 F_1 数学模型

特征表示场 F_1 由三层神经元构成，底层接收来自外界的输入，顶层接收来自 F_2 的外星反馈输入，在中间层对这两种输入进行相应的转换、比较并保存结果，将输出返回顶层神经元及底层神经元。

输入模式 \mathbf{X} 是一个 M 维模拟向量，表示为：

$$\mathbf{X} = (x_1, x_2, \dots, x_M)$$

在 F_1 中有相应的 M 个处理单元，每个单元都包括上、中、下三层，每层都包含两种不同功能的神经元，一种用小空心圆表示，另一种用大实心圆表示，它们的功能分别为：

(1) 空心圆神经元：每个空心圆代表的神经元有两种输入激励，一种是兴奋激励，代表指向神经元的特定模式；一种是抑制激励，代表增益控制输入。设神经元 i 的输出用 V_i 表示，所有兴奋激励的总和为 J_i^+ ，所有抑制的总和为 J_i^- ，则 F_1 的 STM 方程为：

$$\varepsilon \frac{dV_i}{dt} = -AV_i + (1 - BV_i)J_i^+ - (C + DV_i)J_i^- \quad i = 1, 2, \dots, M \quad (15-1)$$

式中， ε 和 A 都是远小于 1 的正实数，且 $\varepsilon < A$ ， $B < 1$ ， $C < D$ ， D 接近于 1。如果 $B = C = 0$ ，且 $\varepsilon \rightarrow 0$ ，式 (15-1) 可简化为：

$$V_i = J_i^+ / (A + DJ_i^-) \quad i = 1, 2, \dots, M \quad (15-2)$$

(2) 实心圆神经元：实心圆神经元的功能是求输入向量的模。在图 15-6 中， F_1 的底层和中层构成一个闭合的正反馈回路，其中标记为 z_i 的神经元接收输入信号 x_i ，而标记为 v_i 的神经元接收上层送来的信号 $bf(s_i)$ 。这个回路中还包括两次规格化运算和一次非线性变换，其中底层输入方程和规格化运算为：

$$\begin{aligned} z_i &= x_i + au_i \\ q_i &= z_i / (e + \|Z\|) \end{aligned} \quad (15-3)$$

式中， e 为很小的正实数，相对于 $\|Z\|$ 可以忽略不计。

中层输入方程和规格化运算为：

$$\begin{aligned} v_i &= f(q_i) + bf(s_i) \\ u_i &= v_i / (e + \|V\|) \end{aligned} \quad (15-4)$$

式中， e 为很小的正实数，相对于 $\|V\|$ 可以忽略不计。

底层至中层和中层至上层之间的非线性变换函数 $f(x)$ 可以采用如下两种形式：

$$f(x) = \begin{cases} 2\theta x^2 / (x^2 + \theta^2) & 0 \leq x < \theta \\ x & x \geq \theta \end{cases} \quad (15-5)$$



$$f(x) \begin{cases} 0 & 0 \leq x < \theta \\ x & x \geq \theta \end{cases} \quad (15-6)$$

式中, a, b 和 θ 由实验而定。

F_1 的中层和上层也构成一个闭合正反馈回路, 其中标记为 p_i 的神经元接收来自中层的信号 u_i 和来自 F_2 场的信号, 这个回路包括的运算是:

$$s_i = p_i / (e + \|P\|) \quad (15-7)$$

$$p_i = u_i + \sum_{j=M+1}^N g(y_j) z_{ji} \quad (15-8)$$

式 (15-7) 中 e 为很小的正实数, 相对于 $\|P\|$ 可以忽略不计。式 (15-8) 中第二项是 F_2 场对神经元 p_i 的输入, z_{ji} 是自上而下的 LTM 系数。

2) 类别表示场 F_2 的数学模型

类别表示场 F_2 的作用是增强自下而上 ($F_1 \rightarrow F_2$) 的滤波输入模式的对比度, 对比度的增强是通过 F_2 的竞争实现的。设 F_2 场中第 j 个神经元的输入为:

$$T_j = \sum_{i=1}^M p_i z_{ji} \quad j = M+1, \dots, N \quad (15-9)$$

F_2 按式 (15-10) 进行选择:

$$T_{j^*} = \max\{T_j\} \quad j = M+1, \dots, N \quad (15-10)$$

当选择神经元 j^* 为最大激活时, 其余神经元处于抑制状态。主要元素为:

$$g(y_j) = \begin{cases} d & j = j^* \\ 0 & j \neq j^* \end{cases} \quad (15-11)$$

式中, d 为自上而下 ($F_1 \rightarrow F_2$) 的反馈参数, $0 < d < 1$ 。式 (15-11) 可简化为:

$$p_i = \begin{cases} u_i + dz_{ij} & j = j^* \\ u_i & j \neq j^* \end{cases} \quad (15-12)$$

3) 权值调整规则——LTM 方程

对长期记忆 LTM 权值的调整, 按以下两个 LTM 方程进行。自上而下 ($F_1 \rightarrow F_2$) 的 LTM 方程为:

$$\frac{dz_{ji}}{dt} = g(y_j)(p_j - z_{ji}) \quad (15-13)$$

自下而上 ($F_2 \rightarrow F_1$) LTM 方程为

$$\frac{dz_{ij}}{dt} = g(y_j)(p_j - z_{ij}) \quad (15-14)$$

当 F_2 确定获胜神经元 j^* 后, 对于 $j \neq j^*$, 有

$$\frac{dz_{ji}}{dt} = 0, \quad \frac{dz_{ij}}{dt} = 0$$

当 $j = j^*$ 时, 则有



$$\frac{dz_{j^*i}}{dt} = d(p_i - z_{j^*i}) = d(1-d) \left(\frac{u_i}{1-d} - z_{j^*i} \right) \quad (15-15)$$

$$\frac{dz_{ij^*}}{dt} = d(p_i - z_{ij^*}) = d(1-d) \left(\frac{u_i}{1-d} - z_{ij^*} \right) \quad (15-16)$$

初始化时, 可取 $z_{ij} = 0$, $z_{ij} = \frac{1}{1(1-d)\sqrt{M}}$, $i = 1, 2, \dots, M$; $j = M+1, M+2, \dots, M+N$, $d = 0.9$ 。

4) 取向子系统

图 15-6 中左侧为取向子系统, 其功能是根据 F_1 的短期记忆模式与激活节点的长期记忆模式之间的匹配度决定 F_2 的重置。匹配度定义为:

$$r_i = \frac{u_i + cp_i}{e + \|U\| + \|cP\|} \quad i = 1, 2, \dots, M \quad (15-17)$$

式中 e 可忽略。实心圆 A 的输出为匹配度的模, 用 $\|R\|$ 表示。设警戒门限为 ρ , $0 < \rho < 1$, 当 $\|R\| > \rho$ 时, 选中该类; 否则, 取向子系统需对 F_2 重置。

15.2.3 ART-2 型网络在系统辨识中的应用

控制系统中常将被控对象看作二阶系统, 其性能可用从单位阶跃响应曲线中抽取的特征参数如上升时间、调整时间、超调量等来描述。一组特征参数构成的特征向量即代表一个二阶对象。ART-2 对二阶系统的辨识实际上是对其特征向量进行分类, 系统辨识的实施方案如图 15-7 所示。其中系统模拟器用来对各种二阶系统的单位阶跃响应进行仿真, 特征抽取器从仿真曲线中提取了 6 个特征参数, 送入 ART-2 网作为输入模式。ART-2 网对输入模式向量的分类是一种有导师学习方式, 其中导师信号来自系统模拟器所仿真的二阶系统数学模型中的参数。二阶系统传递函数的标准形式可写为:

$$G(s) = \frac{\omega_0^2}{s^2 + 2\xi\omega_0 s + \omega_0^2}$$

式中, 无阻尼振荡频率 ω_0 和阻尼比 ξ 可唯一确定二阶系统的传递函数。两个系统参数的不同组合可确定多种二阶系统, 每个二阶系统可对应于一组从阶跃响应曲线中抽取的特征参数。

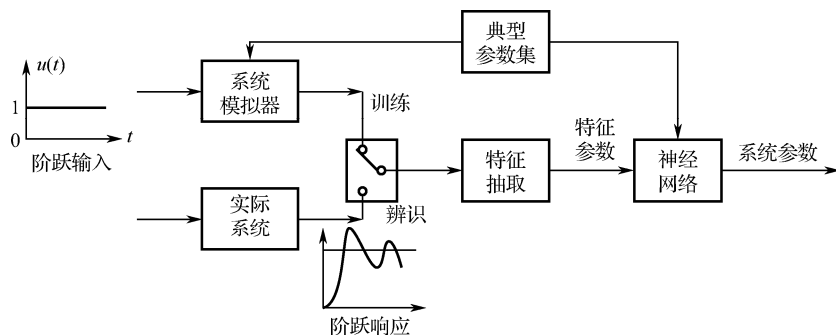


图 15-7 ART-2 网系统辨识器



设 $\xi \in [0.3, 1.3]$, $\omega_0 \in [1.5, 2.0]$, 在该取值范围内列出 17 种传递函数模式供系统模拟器产生阶跃响应曲线。因此该 ART-2 网系统辨识器的 F_1 场有 6 个神经元, F_2 场有 17 个神经元。将训练集中的模式依次输入网络进行训练, 网络根据输入的特征向量修改相应的 LTM 权值, 并在 F_2 场指定一神经元作为该模式的代表。训练结束后 F_2 场的每个神经元即代表一种系统特征模式, 当该系统辨识器实际使用时, 来自实际系统的阶跃响应曲线经过特征抽取器后输入系统辨识器激活相应的神经元, 复位子系统对该模式与存储模式类的相似性进行检查, 如大于警戒门限则将其归类, 否则指定一个新神经元代表该模式类。

警戒门限的大小对分类的粗细具有调节作用, 本例将其设为 0.99, 以保证分类具有较高的分辨力。

15.3 CP 神经网络概述

人工神经网络或称连接机制, 是由简单信息处理单元(神经元)互连组成的网络, 能接收并处理信息, 它是通过把问题表达成处理单元之间的连接权来处理的。

对向传播(Counter-Propagation, CP)神经网络是将 KohonenSOM 网络与 Grossberg 基本竞争型网络相结合, 发挥各自特长的一种新型 SOM 网络。这一网络是美国神经计算机专家 Robert Hecht-Nielsen 于 1987 年提出的。这一网络有效地应用于模式分类、函数近似、统计分析和数据压缩等领域。网络结构如图 15-8 所示。网络分输入层、竞争层、输出层三层。输入层与竞争层构成 SOM 网络, 竞争层与输出层构成基本竞争型网络。网络从整体上看属于有教师示教型网络, 而由输入层和竞争层构成的 SOM 网络又是一种典型的无教师示教型网络。因此, 这一网络既吸取了无教师示教型网络分类灵活、算法简练的优点, 又采纳了有教师示教型网络分类精细、准确的长处, 使两种不同类型的网络有机地结合起来。

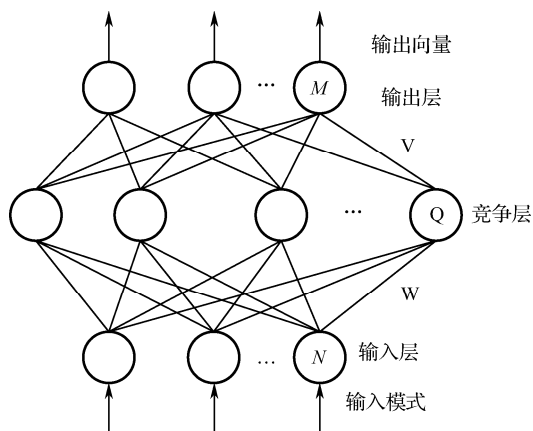


图 15-8 CP 网络结构

由输入层至竞争层, 网络按 SOM 学习规则产生竞争层的获胜神经元, 并按这一规则调整相应的输入层至竞争层的连接权; 由竞争层至输出层, 网络按基本竞争型网络学习规则, 得到各输出神经元的实际输出值, 并按有教师示教的误差校正方法, 调整由竞争层至输出层的



连接权。经过这样反复学习，可以将任意输入模式映射为输出模式。由 CP 网络的这一基本思想可以发现，处于网络中间位置的竞争层获胜神经元及与其相关是连接权向量，既反映了输入模式的统计特征，又反映了输出模式的统计特征。因此，可以认为输入、输出模式通过竞争实现了相互映射，即网络具有双向记忆功能

15.3.1 CP 网络学习

假定输入层有 N 个神经元， p 个连续值的输入模式为 $\mathbf{A}_k = (a_1^k, a_2^k, \dots, a_N^k)$ ，竞争层有 Q 个神经元，对应的二值输出向量为 $\mathbf{B}_k = (b_1^k, b_2^k, \dots, b_Q^k)$ ，输出层有 M 个神经元，其连续值的输出向量为 $\mathbf{C}'_k = (c_1'^k, c_2'^k, \dots, c_M'^k)$ ，目标输出向量为 $\mathbf{C}_k = (c_1^k, c_2^k, \dots, c_M^k)$ ，上述 k 值为 $k=1, 2, \dots, p$ 。由输入层到竞争层的连续权值向量为 $\mathbf{W}_j = (w_{j1}, w_{j2}, \dots, w_{jN})$ ， $j=1, 2, \dots, Q$ ；由竞争层到输出层的连接权向量为 $\mathbf{V}_l = (v_{l1}, v_{l2}, \dots, v_{lQ})$ ， $l=1, 2, \dots, M$ 。网络学习和工作规则如下所述。

(1) 初始化。将连接权向量 \mathbf{W}_j 和 \mathbf{V}_l 赋予区间[0,1]内的随机值。将所有的输入模式 \mathbf{A}_k 进行归一化处理：

$$a_i^k = \frac{a_i^k}{\|\mathbf{A}_k\|}$$

$$\|\mathbf{A}_k\| = \sqrt{\sum_{i=1}^N (a_i^k)^2}, i=1, 2, \dots, N$$

(2) 将第 k 个输入模式 \mathbf{A}_k 提供给网络的输入层。

(3) 将连接权值向量 \mathbf{W}_j 按照式 (15-18) 进行归一化处理：

$$w_{ji} = \frac{w_{ji}}{\|\mathbf{w}_{ji}\|}$$

$$\|\mathbf{w}_{ji}\| = \sqrt{\sum_{i=1}^N w_{ji}^2}, i=1, 2, \dots, N \quad (15-18)$$

(4) 竞争层中每个神经元的加权输入和：

$$S_j = \sum_{i=1}^N a_i^k w_{ji}, j=1, 2, \dots, Q$$

(5) 求连接权向量 \mathbf{W}_j 中与 \mathbf{A}_k 距离最近的向量 \mathbf{W}_g ：

$$\mathbf{W}_g = \max_{j=1, 2, \dots, Q} \sum_{i=1}^N a_i^k w_{ji} = \max_{j=1, 2, \dots, Q} S_j$$

将神经元 g 的输出设定为 1，其余竞争层神经元的输出设定为 0：

$$b = \begin{cases} 1, & j = g \\ 0, & j \neq g \end{cases}$$

(6) 将连接权向量 \mathbf{W}_g 按照式 (15-19) 进行修正：

$$w_{gi}(t+1) = w_{gi}(t) + \alpha(a_i^k - w_{gi}(t)); i=1, 2, \dots, N \quad (15-19)$$



其中， $0 \leq \alpha \leq 1$ 为学习率。

(7) 将连接权向量 W_g 重新归一化，归一化算法同上。

(8) 按照式 (15-20) 修正竞争层到输出层的连接权向量 V_l ：

$$v_{li}(t+1) = v_{li}(t) + \beta b_j(c_l - c_j); \quad l=1,2,\dots,M, j=1,2,\dots,Q \tag{15-20}$$

其中， $0 \leq \beta \leq 1$ 为学习率。由 (5) 可将上式简化为：

$$v_{lg}(t+1) = v_{lg}(t) + \beta b_j(c_l - c_j), l=1,2,\dots,M$$

由此可见，只需要调整竞争层获胜神经元 g 到输出层神经元的连接权向量 V_g 即可，其他连接权向量保持不变。

(9) 求输出层各神经元的加权输入，并将其作为输出神经元的实际输出值， $c'_l = \sum_{j=1}^Q b_j v_{lg}$ ，

$l=1,2,\dots,M$ ，同时可将其简化为 $c'_l = v_{lg}$ 。

(10) 返回 (2)，直到将 P 个输入模式全部提供给网络。

(11) 令 $t=t+1$ ，将输入模式 A_k 重新提供给网络学习，直到 $t=T$ ，其中 T 为预先设定的学习总次数，一般取 $500 < T < 10000$ 。

15.3.2 CP 网络应用

这里举一个非常简单而且与日常生活相关的例子来说明 CP 网络的应用。

【例 15-2】 现在需要创建一个 CP 网络，其任务是在已知一个人本星期应该完成的工作量和此人当时的思想状态的情况下，对此人星期日下午的活动安排提出建议。

按照一般情况，将工作量分为 3 挡，即没有、有一些和很多，所对应的量化值分别为 0.0、0.5 和 1.0；把思想情绪也分为 3 个水平，即低、一般和高，所对应的量化值分别为 0.0、0.5 和 1.0。可选择的活动有 5 个，即在家里看画报、去商场购物、到公园散步、与朋友一起吃饭和干工作。工作量和思想情绪状态一共有 6 种组合，这 6 种组合分别对应各自的最佳活动选择，样本模式如表 15-1 所示。

表 15-1 网络训练样本模式

工 作 量	思 想 情 绪	活 动 安 排	目 标 输 出
没有 0.0	低 0.0	看画报	10000
有一些 0.5	低 0.0	看画报	10000
没有 0.0	一般 0.5	购物	01000
很多 1.0	高 1.0	公园散步	00100
有一些 0.5	高 1.0	吃饭	00010
很多 1.0	一般 0.5	工作	00001

把这组训练样本提供给网络进行充分学习后，网络就具有了一种“内插”功能，即当网络输入一对在 (0,1) 区间中反映工作量和情绪的量化值后，网络将自动根据原有的记忆，找出对应于这对量化值的最佳活动选择，以输出模式的形式提供给用户作为决策参数。



实际上,不光 CP 网络具有这种“内插”功能,BP 网络、SOM 网络都具有这种功能。从模式识别的角度上讲,这些网络具有对输入模式进行分类的功能。

可惜的是,对功能如此强大的 CP 网络,神经网络工具箱中竟然没有为之支持的函数工具。但是,既然 15.3.1 节中已经给出了有关 CP 的学习和训练算法过程,因此,我们可以利用 MATLAB 强大的数学计算功能,实现解决该问题的 CP 网络。

根据题意,该网络的输入层应该有 2 个神经元,输出层应该有 5 个神经元。为了更加准确地解决问题,将竞争层神经元设置为 18 个。网络结构如图 15-9 所示。

由表 15-1 可得,网络的输入向量为:

```
P=[0 0;0.5 0.5;0 0.5;1 1;0.5 1;1 0.5];
```

目标向量为:

```
T=[1 0 0 0 0;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1];
```

下面对网络进行一个周期的学习。令输入层和竞争层之间的连接权向量矩阵用 W 表示,竞争层和输出层之间的权向量矩阵用 V 表示。可知 W 为一个 18×2 的矩阵, V 是一个 5×18 的矩阵,学习速率设定为 0.1。

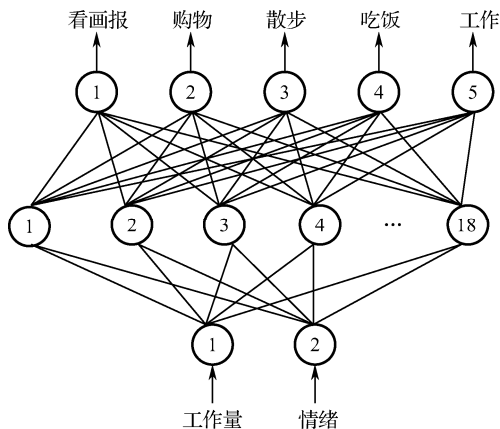


图 15-9 星期日下午活动安排决策 CP 网络

(1) 初始化。利用 MATLAB 中的随机数产生函数 W 和 V , 并赋区间 $[0,1]$ 之间的随机值, 代码如下:

```
W=rands(18,2)/2+0.5;
V=rands(5,18)/2+0.5;
```

由于函数 rands 产生的随机数位于区间 $(-1,1)$ 之间, 所以这里做了这样的处理, 使得产生的随机数既不影响随机性能, 又位于区间 $[0,1]$ 中。

对输入向量进行归一化处理:

```
W=rands(18,2)/2+0.5;
V=rands(5,18)/2+0.5;
for i=1:6
    if(P(i,:)==[0 0])
        P(i,:)=P(i,:)
```



```

else
    P(i,:)=P(i,:)/norm(P(i,:));
end
end

```

之所以要在循环中进行数据判断，是因为向量[0 0]是无法归一化处理的，比如 P 的第一组元素就是正在用的这一组中。

(2) 将第一个输入样本 (0 0) 提供给网络的输入层神经元。

(3) 对连接权向量 W 进行归一化处理。

(4) 求每一个竞争层神经元的加权输入 s_j , $j = 1, 2, \dots, 18$:

```

for i=1:18
    W(i,:)=W(i,:)/norm(W(i,:));
    s(i)=P(1,:)*W(i,:);
end

```

循环语句中第一句用于对连接权向量 W 进行归一化处理，第二句可以求出竞争层每个神经元的输出。结果为：

```

s =
    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0

```

(5) 求连接权向量 W 中与 (0 0) 距离最近的向量，由于输出全部为 0，所以可任选一个权值向量 W_g ，这里选 18，并将该神经元的输出设定为 1。

```

for i=1:18
    W(i,:)=W(i,:)/norm(W(i,:));
    s(i)=P(1,:)*W(i,:);
end
temp=max(s);
for i=1:18
    if temp==s(i)
        count=i;
    end
end
%将所有竞争层神经元的输出置为 0
for i=1:18
    s(i)=0;
end
%选中的神经元输出为 1
s(count)=1;

```

(6) 调整连接权向量 W_{18} ，并重新将其归一化。

```

W(count,:)=W(count,:)+0.1*[P(1,:)-W(count,:)];
W(count,:)=W(count,:)/norm(W(count,:))

```



输出结果为:

```
W(18,:)=
    0.9997    0.0251
```

经检验,此时的 W18 确实已经归一化了。

(7) 调整竞争层神经元到输出层神经元之间的连接权向量 V:

```
V(:,count)=V(:,count)+0.1*(T(1,:)-T_out(1,:));
```

由于此时输出层神经元的输出域目标向量是一致的,所以这里的权值是没有经过调整的,等来了下一个模式后,权值才会真正得到调整。

(8) 计算输出层各神经元的加权输入,并将其作为神经元的实际输出值:

```
T_out(1,:)=V(:,count)';
```

第一组实际输出就等于竞争层中第 18 个神经元与输出层各神经元之间调整后的连接权值。

(9) 返回步骤(2),将输入向量中的(0.5 0.5)提供给网络。

(10) 继续学习,直到训练次数达到设定的最大值。

CP 网络训练结束后,按照以下步骤进行网络回想。

(1) 将输入模式 A 提供给网络的输入层。

(2) 根据式(15-21)求出竞争层的获胜神经元 g 。

$$b_g = \max_{i=1,2,\dots,Q} \left(\sum_{i=1}^N w_{ji} a_i \right) \quad (15-21)$$

(3) 令 $b_g = 1$, 其余的输出都等于 0。按照式(15-22)求得输出层各神经元的输出。

$$c_j = v_{jg} b_g \quad (15-22)$$

由此产生了输出模式 $C = (c_1, c_2, \dots, c_M)$, 从而就得到了输入 A 的分类结果。

实现本例的完整 MATLAB 代码如下:

```
clear all
%初始化正向权值 W 和反向权值 V
W=rands(18,2)/2+0.5;
V=rands(5,18)/2+0.5;
%输入向量 P 和目标向量 T
P=[0 0;0.5 0.5;0 0.5;1 3;0.5 1;1 0.5];
T=[1 0 0 0 0;1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0;0 0 0 0 1];
T_out=T;
%设定学习步数为 1000 次
epoch=1000;
%归一化输入向量 P
for i=1:6
    if P(i,:)==[0 0]
        P(i,:)=P(i,:);
    else
```



```

        P(i,:)=P(i,:)/norm(P(i,:));
    end
end
%开始训练
while epoch>0
    for j=1:6
        %归一化正向权值 W
        for i=1:18
            W(i,:)=W(i,:)/norm(W(i,:));
            s(i)=P(j,:)*W(i,:);
        end
        %求输出最大的神经元, 即获胜神经元
    temp=max(s);
    for i=1:18
        if temp==s(i)
            count=i;
        end
    end
    %将所有竞争层神经元的输出置为 0
    for i=1:18
        s(i)=0;
    end
    %选中的神经元输出为 1
    s(count)=1;
    %权值调整
    W(count,:)=W(count,:)+0.1*[P(j,:)-W(count,:)];
    W(count,:)=W(count,:)/norm(W(count,:));
    V(:,count)=V(:,count)+0.1*(T(j,:)-T_out(j,:));
    %计算网络输出
    T_out(j,:)=V(:,count)';
    %end
    %训练次数递减
    epoch=epoch - 1;
end
%训练结束
T_out
%网络回想
%网络的输入模式 Pc
Pc=[0.5 1;1 3];
%初始化 Pc
for i=1:2
    if Pc(i,:)==[0 0]

```





```

        Pc(i,:)=Pc(i,:);
    else
        Pc(i,:)=P(i,:)/norm(Pc(i,:));
    end
end
%网络输出
Outc=[0 0 0 0 0;0 0 0 0 0];
for j=1:2
    for i=1:18
        sc(i)=Pc(j,:)*W(i,:);
    end
    tempc=max(sc);
    for i=1:18
        if tempc==sc(i)
            countp=i;
        end
        sc(i)=0;
    end
    sc(countp)=1;
    Outc(j,:)=V(:,countp)';
end
%回想结束
Outc

```

输出结果为:

```

T_out =
    1    1    0    0    0    0
    0    0    1    0    0    0
    0    0    0    1    0    0
    0    0    0    0    1    0
    0    0    0    0    0    1

```

由此可见, 经过 1000 次训练后, 网络的实际输出和目标输出就一致了, 这说明训练过程是有效的。

```

Outc =
    0.3050    0.8744    0.0150    0.7680    0.9708
    0.3784    0.8600    0.8537    0.5936    0.4966

```

Outc 是网络回想的输出, 实际上也就是网络测试的结果, 在这里给出了两种特定的组合状态, 即 (0.5 1) 和 (1 1) 的组合, 这两种组合分别对应吃饭和到公园散步, 可见, 网络给出了正确的建议。

第 16 章 Hopfield 网络算法分析与实现



在多输入、多输出的动态系统中，控制对象特性复杂，传统方法难以描述复杂的系统。为控制对象建立模型，可以减少直接进行实验带来的负面影响，所以模型显得尤为重要。但是，前馈神经网络从结构上说属于一种静态网络，其输入、输出向量之间是简单的非线性函数映射关系。实际应用中系统过程大多是动态的，前馈神经网络辨识就暴露出明显的不足，用前馈神经网络只是非线性对应网络，无反馈记忆环节。为了克服前馈神经网络的缺点，使神经网络更加接近系统的实际过程，利用反馈神经网络的动态特性，拥有记忆环节，对一个多输入、多输出复杂的对象进行辨识，测试了建模效果并与实验效果进行了对比。

16.1 Hopfield 神经网络

Hopfield 网络是神经网络发展历史上的一个重要的里程碑。由美国加州理工学院物理学家 J.J.Hopfield 教授于 1982 年提出，是一种单层反馈神经网络。

Hopfield 网络是一种由非线性元件构成的反馈系统，其稳定状态的分析比前向神经网络要复杂得多。1984 年，Hopfield 设计并研制了网络模型的电路，并成功地解决了旅行商（TSP）的计算难题（优化问题）。

Hopfield 网络分为离散型和连续型两种网络模型，分别记作 DHNN（Discrete Hopfield Neural Network）和 CHNN（Continues Hopfield Neural Network）。

16.1.1 离散型 Hopfield 网络

在离散 Hopfield 网络中，所采用的神经元是二值神经元；因此，所输出的离散值 1 和 0 分别表示神经元处于激活和抑制状态。DHNN 网络的拓扑结构如图 16-1 所示。

1. 网络的状态

DHNN 网中的每个神经元都有相同的功能，其输出称为状态，用 x_j 表示，所有神经元状态的集合就构成反馈网络的状态 $X = (x_1, x_2, \dots, x_n)^T$ 。反馈网络的输入就是网络的状态初始值，表示为 $X(0) = (x_1(0), x_2(0), \dots, x_n(0))^T$ 。反馈网络在外界输入激发下，从初始状态进入动态演变过程，其间网络中每个神经元的状态在不断变化，变化规律由下式规定：

$$x_j = f(\text{net}_j) \quad j = 1, 2, \dots, n$$

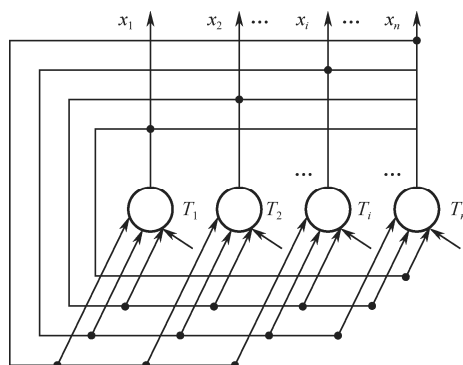


图 16-1 DHNN 网的拓扑结构

其中, $f(\cdot)$ 为转移函数, DHNN 网的转移函数常采用符号函数:

$$x_j = \text{sgn}(\text{net}_j) = \begin{cases} 1 & \text{net}_j \geq 0 \\ -1 & \text{net}_j < 0 \end{cases} \quad j = 1, 2, \dots, n \quad (16-1)$$

式中净输入为:

$$\text{net}_j = \sum_{i=1}^n (w_{ij}x_i - T_j) \quad j = 1, 2, \dots, n \quad (16-2)$$

对于 DHNN 网, 一般有 $w_{ii} = 0, w_{ij} = w_{jw}$ 。

反馈网络稳定时每个神经元状态都不再改变, 此时的稳定状态就是网络的输出, 表示为:

$$\lim_{i \rightarrow \infty} X(t)$$

2. 网络工作方式

DHNN 网络中工作方式分为异步工作方式及同步工作方式。

1) 网络的同步工作方式

网络的同步工作方式是一种并行方式, 所有神经元同时调整状态, 即

$$x_j(t+1) = \text{sgn}[\text{net}_j(t)] \quad j = 1, 2, \dots, n \quad (16-3)$$

2) 网络的异步工作方式

网络的异步工作方式是一种串行方式。网络运行时每次只有一个神经元 i 按式 (16-1) 进行状态的调整计算, 其他神经元的状态均保持不变, 即

$$x_j(t+1) = \begin{cases} \text{sgn}[\text{net}_j(t)] & j = i \\ x_j(t) & j \neq i \end{cases} \quad (16-4)$$

神经元状态的调整次序可以按某种规定的次序进行, 也可以随机选定。每次神经元在调整状态时, 根据其当前净输入正负决定下一时刻的状态, 因此其状态可能会发生变化, 也可能保持原状。下次调用其他神经元状态时, 本次的调整结果即在下一个神经元的净输入中发挥作用。



3. 网络结构

DHNN 是一种单层的、其输入/输出为二值的反馈网络。假设有一个由三个神经元组成的离散 Hopfield 神经网络，其结构如图 16-2 所示。

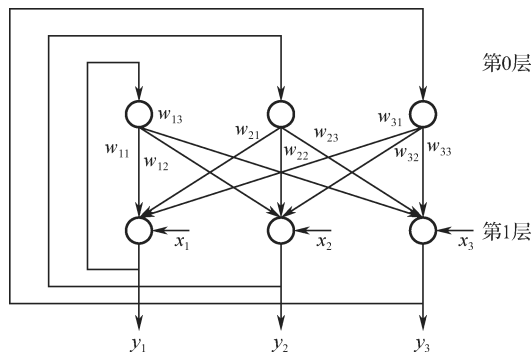


图 16-2 离散 Hopfield 网络图示

在图 16-2 中，第 0 层仅仅是作为网络的输入，它不是实际神经元，所以无计算功能；第一层是神经元，故而执行对输入信息和权系数乘积求累加和，并经非线性函数 f 处理后产生输出信息。 f 是一个简单的阈值函数，如果神经元的输出信息大于阈值 θ ，那么，神经元的输出取值为 1；小于阈值 θ ，则神经元的输出取值为 θ 。

对于二值神经元，它的计算公式如下：

$$u_j = \sum_i w_{ij} y_i + x_j \quad (16-5)$$

其中， x_j 为外部输入，并且有：

$$\begin{cases} y_i = 1, & u_i \geq \theta_i \\ y_i = 0, & u_i < \theta_i \end{cases} \quad (16-6)$$

一个 DHNN 的网络状态是输出神经元信息的集合。对于一个输出层是 n 个神经元的网络，其 t 时刻的状态为一个 n 维向量：

$$Y(t) = [y_1(t), y_2(t), \dots, y_n(t)]^T \quad (16-7)$$

因为 $y_i(t) (i=1, 2, \dots, n)$ 可以取值为 1 或 0，故 n 维向量 $Y(t)$ 有 2^n 种状态，即网络有 2^n 种状态。

对于三个神经元的 DHNN，它的输出层就是三位二进制数；每一个三位二进制数就是一种网络状态，共有 8 个网络状态，这些网络状态如图 16-3 所示。在图中，立方体的每一个顶角表示一种网络状态。同理，对于 n 个神经元的输出层，它有 2^n 种网络状态，也和一个 n 维超立方体的顶角相对应。

如果 Hopfield 网络是一个稳定网络，若在网络的输入端加入一个输入向量，则网络的状态会产生变化，即从超立方体的一个顶角转向另一个顶角，并且最终稳定于一个特定的顶角。

对于一个 n 个神经元组成的 DHNN，则有 $n \times n$ 权系数矩阵 W ：

$$W = \{w_{ij}\}, \quad i = 1, 2, \dots, n; j = 1, 2, \dots, n \quad (16-8)$$

同时，有 n 维阈值向量 θ ：



$$\theta = [\theta_1, \theta_2, \dots, \theta_n]^T \quad (16-9)$$

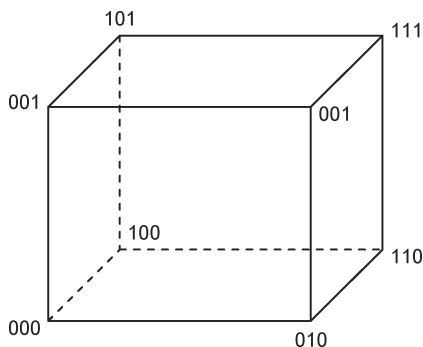


图 16-3 三神经元 Hopfield 网络输出层的网络状态图

一般而言, W 和 θ 可以确定一个唯一的 DHNN。对于如图 16-2 所示的三神经元组成的 Hopfield 网络, 也可以改用如图 16-4 所示的图形表示, 这两个图形的意义是一样的。

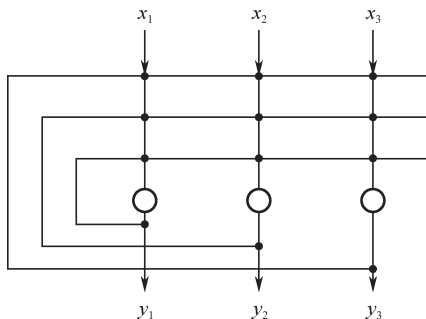


图 16-4 DHNN 的另外一种图示

考虑 DHNN 的一般节点状态, 用 $y_j(t)$ 表示第 j 个神经元, 即节点 j 在时刻 t 的状态, 则节点的下一个时刻 $t+1$ 的状态可以求得如下:

$$y_j(t+1) = f[u_j(t)] = \begin{cases} 1, & u_j \geq 0 \\ 0, & u_j < 0 \end{cases} \quad (16-10)$$

$$u_j(t) = \sum_{i=1}^n w_{ij} y_i(t) + x_j - \theta_j$$

如果 w_{ij} 在 $i=j$ 时等于 0, 说明一个神经元的输出并不会反馈到其输入端, 这时, DHNN 称为无自反馈的网络; 如果 w_{ij} 在 $i=j$ 时不等于 0, 说明一个神经元的输出会反馈到其输入端, 这时, DHNN 称为自反馈的网络。

16.1.2 DHNN 的动力学稳定性

对 HNN 的能量函数有几点说明:

(1) 当对反馈网络应用能量函数后, 从任一初始状态开始, 因为在每次迭代后都能满足 $\Delta E = 0$, 所以网络的能量将会越来越小。



由于能量函数存在下界,因此其最后趋于稳定点 $\Delta E = 0$

(2) Hopfield 能量函数的物理意义为:

- 在那些渐进稳定点的吸引域内, 离吸引点越远的状态, 所具有的能量越大。
- 由于能量函数的单调下降特性, 保证状态的运动方向能从远离吸引点处不断地趋向于吸引点, 直到达到稳定点。

(3) 能量函数是反馈网络中的重要概念。根据能量函数可以方便地判断系统的稳定性。

(4) Hopfield 选择的能量函数只是保证系统稳定和渐进稳定的充分条件, 而不是必要条件, 其能量函数也不是唯一的。

(5) 在状态更新过程中, 包括三种情况: 由 0 变为 1; 由 1 变为 0 及状态保持不变。

类似于研究动力学系统稳定性的 Lyapunov 稳定性理论, 上述 DHNN 的稳定性可由分析上述定义的 Lyapunov 函数 E 的变化规律揭示。

因此, 由神经元 j 的状态变化量 $\Delta y_j(t)$ 所引起的能量变化量 ΔE_j 为:

$$\Delta E_j = \frac{\partial E}{\partial y_j(t)} \Delta y_j(t) = \left[-\frac{1}{2} \sum_{i=1}^n (w_{i,j} + w_{j,i}) y_i(t) - x_j + \theta_j \right] \Delta y_j(t) \quad (16-11)$$

如果所讨论的 HNN 是对称网络, 即有 $w_{i,j} = w_{j,i}$, $i, j = 1, 2, \dots, n$ 。

$$\Delta E_j = \left[-\sum_{i=1}^n w_{i,j} y_i(t) - x_j + \theta_j \right] \Delta y_j(t) \quad (16-12)$$

如果, 令

$$u_j(t) = \sum_{i=1}^n w_{i,j} y_i(t) + x_j \quad (16-13)$$

则

$$y_i(t+1) = f[u_i(t) - \theta_i]$$

式 (16-13) 可记为:

$$\Delta E_j(t) = [-u_j(t) + \theta_j] \Delta y_j(t) \quad (16-14)$$

下面分别对异步方式和同步方式证明对称二值型 HNN 是稳定的。

1) 异步方式

对串行异步和对称权值型的 HNN, 基于式 (16-14) 考虑如下两种情况:

第一, 如果 $u_j > \theta_j$, 即神经元 j 的输入综合大于阈值, 则从二值神经元的计算公式知道:

- y_j 的值保持为 1, 或者从 0 变到 1。
- 这说明 y_j 的变化 Δy_j 只能是 0 或正值, 这时很明显有:

$$\Delta E_j \leq 0$$

这说明 HNN 神经元的能量减少或不变。

第二, 如果 $u_j < \theta_j$, 即神经元 j 的输入综合小于阈值, 则知 y_j 的值保持为 0, 或者从 1 变到 0, 而 Δy_j 小于等于零, 这时则有 ΔE_j :

$$\Delta E_j \leq 0$$

这也说明 HNN 神经元的能量减少。

上面两点说明了 DHNN 在权系数矩阵 W 的对角线元素为 0, 而且 W 矩阵元素对称时, 串行异步方式的 DHNN 是稳定的。

2) 并行同步方式

由上述对串行异步和对称权值型 DHNN 的稳定性分析过程可知, 单个神经元的状态变化引起的 Lyapunov 函数的变化量:

$$\Delta E_j(t) \leq 0$$

因此, 并行同步且权值对称的 DHNN 的所有神经元引起的 Lyapunov 函数的变化量为:

$$\Delta E = \sum_{j=1}^n \frac{\partial E}{\partial y_j(t)} \Delta y_j(t) = \sum_{j=1}^n \Delta E_j(t) \leq 0 \quad (16-15)$$

故上面两点说明了 DHNN 在权系数矩阵 W 的对角线元素为 0, 而且 W 矩阵元素对称时, 并行同步方式的 DHNN 是稳定的。

基于上述分析, Coben 和 Grossberg 在 1983 年给出了关于 HNN 稳定的充分条件, 他们指出:

- 如果权系数矩阵 W 是一个对称矩阵, 并且, 对角线元素为 0, 则这个网络是稳定的。

- 即在权系数矩阵 W 中, 若:

$i = j$ 时, $w_{ij} = 0$

$i \neq j$ 时, $w_{ij} = w_{ji}$

则 HNN 是稳定的。

应该指出: 这只是 HNN 稳定的充分条件, 而不是必要条件。

在实际中有很多稳定的 HNN, 但是它们并不满足权系数矩阵 W 是对称矩阵这一条件。

由以上的分析可知: 无自反馈的权系数对称 HNN 是稳定, 它如图 16-5 所示。

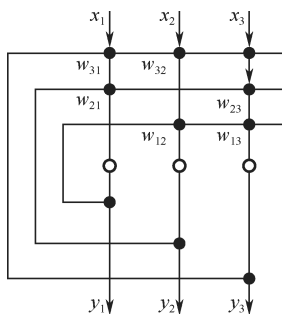


图 16-5 对角线权系数为 0 的对称网图

16.1.3 网络权值的学习

下面通过两种方法来讨论网络权值的学习。

1. 外积型网络权值的学习方法

网络待记忆的学习样本有 N 个, $X^K, K=1, 2, \dots, N$, $X^K \in R^n$, 其每个分量为



$X_i^K, i=1,2,\dots,n$, 利用已知需要存储的样本来设计 n 个节点间的连接权值, 如节点 i 和 j 间的连接权值为:

$$\begin{cases} w_{ij} = \alpha \sum_{K=1}^N X_i^K X_j^K & i \neq j \\ w_{ii} = 0 & i = j \end{cases} \quad (16-16)$$

其中, α 为一个正常数, 初始化时 $w_{ij}=0$, 当每输入一个样本时, 在权值上加修正量 $w_{ij}(t+1) = w_{ij}(t) + \alpha X_i^K X_j^K$, 当第 K 个样本 X_i^K 和 X_j^K 同时兴奋或同时抑制时, $\alpha X_i^K X_j^K > 0$, 当 X_i^K 和 X_j^K 一个兴奋一个抑制时, $\alpha X_i^K X_j^K < 0$ 。用 Hebb 规则修正权值可以满足 $w_{ij} = w_{ji}$ 的条件, 从而使得网络在串行工作方式时保证收敛, 在并行工作时系统或者收敛, 或者出现极限环为 2 的振荡。

把式 (16-16) 写成矩阵的形式, 取 $\alpha=1$, 对于需要记忆的样本 $X^K, K=1,2,\dots,N$, 权值为:

$$\begin{aligned} W &= [X^1 X^2 \dots X^N] \begin{bmatrix} X^{1T} \\ X^{2T} \\ \vdots \\ X^{NT} \end{bmatrix} - NU \\ &= \begin{bmatrix} X_1^1 & X_1^2 & \dots & X_1^K & \dots & X_1^N \\ X_2^1 & X_2^2 & \dots & X_2^K & \dots & X_2^N \\ \vdots & \vdots & & \vdots & & \vdots \\ X_n^1 & X_n^2 & \dots & X_n^K & \dots & X_n^N \end{bmatrix} \begin{bmatrix} X_1^1 & X_1^2 & \dots & X_n^1 \\ X_1^2 & X_2^2 & \dots & X_n^2 \\ \vdots & \vdots & & \vdots \\ X_1^N & X_2^N & \dots & X_n^N \end{bmatrix} - N \begin{bmatrix} 1 & \cdot & \cdot & \cdot & 0 \\ \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \\ 0 & \cdot & \cdot & \cdot & 1 \end{bmatrix} \\ &= \begin{bmatrix} \sum_{K=1}^N (X_1^K)^2 & \sum_{K=1}^N X_1^K X_2^K & \dots & \sum_{K=1}^N X_1^K X_n^K \\ \sum_{K=1}^N X_2^K X_1^K & \sum_{K=1}^N (X_2^K)^2 & \dots & \sum_{K=1}^N X_2^K X_n^K \\ \vdots & \vdots & & \vdots \\ \sum_{K=1}^N X_n^K X_1^K & \dots & \dots & \sum_{K=1}^N (X_n^K)^2 \end{bmatrix} - N \begin{bmatrix} 1 & \cdot & \cdot & \cdot & 0 \\ \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \\ 0 & \cdot & \cdot & \cdot & 1 \end{bmatrix} \end{aligned}$$

由于,

$$\sum_{K=1}^N (X_1^K)^2 = \sum_{K=1}^N (X_2^K)^2 = \dots = \sum_{K=1}^N (X_n^K)^2 = N$$

这里, U 是一个单位矩阵。

所以



$$W = \begin{bmatrix} 0 & \sum_{K=1}^N X_1^K X_2^K & \cdots & \sum_{K=1}^N X_1^K X_n^K \\ \sum_{K=1}^N X_2^K X_1^K & 0 & \cdots & \vdots \\ \vdots & \vdots & & \vdots \\ \sum_{K=1}^N X_n^K X_1^K & \cdots & \cdots & 0 \end{bmatrix} \quad (16-17)$$

因此, Hebb 规则能够满足 $w_{ij} = 0$, $w_{ij} = w_{ji}$ 的条件, 网络能够收敛于稳定点。

2. 稳定点的讨论

可分以下向点对稳定点进行讨论。

(1) 如果待记忆的样本是两两正交, 即对于样本 $X^K, K=1, 2, \dots, N$, X^i 、 X^j 为 X^K 中的任意两个不同的样本, 且满足 $[X^i]^T [X^j] = 0$, $i \neq j$, 对所有的 i 和 j 成立。

在节点的输出为 $X_i \in \{1, -1\}$ 的情况下, 当二个 n 维样本向量的各个分量中有 $n/2$ 个是相同的, 另 $n/2$ 个是相反的, 就能满足这两个向量正交。用上面的外积法所得到的权值进行迭代计算, 在输入样本中任取一个样本 X^K 作为初始输入, 可得:

$$\begin{aligned} WX^K &= [X^1 X^2 \cdots X^K \cdots X^N] \begin{bmatrix} X^{1T} \\ X^{2T} \\ \vdots \\ X^{KT} \\ \vdots \\ X^{NT} \end{bmatrix} X^K - NUX^K \\ &= [X^1 X^2 \cdots X^K \cdots X^N] \begin{bmatrix} 0 \\ 0 \\ \vdots \\ X^{KT} X^K \\ \vdots \\ 0 \end{bmatrix} - NUX^K = nX^K - NX^K = (n - N)X^K \end{aligned} \quad (16-18)$$

只要满足 $n > N$, 则 $\text{sgn}[WX^K] = X^K$, 则 X^K 为网络的一个稳定点。

(2) 如果 N 个样本 $X^K, K=1, 2, \dots, N$, 不是两两正交, 其连接权值依据 Hebb 规则求得, 在 N 个样本中任选一个样本 X^K 作为初始输入:

$$WX^K = [X^1 X^2 \cdots X^K \cdots X^N] \begin{bmatrix} X^{1T} \\ X^{2T} \\ \vdots \\ X^{KT} \\ \vdots \\ X^{NT} \end{bmatrix} X^K - NX^K \quad (16-19)$$



通过式 (16-19) 可求得新的输出 $\text{sgn}(WX^K) = X^{K'}$, 取 $X^{K'}$ 的第 j 个分量:

$$\begin{aligned} X_j^{K'} &= \text{sgn} \left[\sum_{i=1}^n w_i X_i^K \right] = \text{sgn} \left[\sum_{i=1}^n X_i^1 X_j^1 X_i^K + \sum_{i=1}^n X_i^2 X_j^2 X_i^K + \cdots + \sum_{i=1}^n X_i^N X_j^N X_i^K \right] \\ &= \text{sgn} \left[\sum_{i=1}^n X_j^K (X_i^K)^2 + \sum_{k=1, k \neq K}^N \sum_{i=1}^n X_i^k X_j^k X_i^K \right] = \text{sgn}(s_j + n_j) \end{aligned} \quad (16-20)$$

式中, $s_j = nX_j^K$; $n_j = \sum_{k=1, k \neq K}^N \sum_{i=1}^n X_i^k X_j^k X_i^K$ 。

设 n_j 为零均值的随机变量, $X_i^k, X_j^k \in \{1, -1\}$, 而 n_j 的方差 $\sigma^2 = (N-1)n$, $\sigma = \sqrt{(N-1)n}$ 。对于非正交的学习样本, 如果满足 $n > \sqrt{(N-1)n}$, 则网络仍可收敛到其记忆样本上。

16.1.4 联想记忆功能

Hopfield 网络的一个功能是可用于联想记忆, 即联想存储器, 这是人类的智能特点之五。人类的所谓触景生情就是见到一些类同过去接触的景物, 容易产生对过去情景的回味和思忆。对于 Hopfield 网络, 用它作联想记忆时, 首先通过一个学习训练过程确定网络中的权系数, 使所记忆的信息在网络的 n 维超立方体的某一个顶角的能量最小。当网络的权系数确定之后, 只要向网络给出输入向量, 即使这个向量可能是局部数据, 即不完全或部分不正确的数据, 但是网络仍然产生所记忆信息的完整输出。1984 年 Hopfield 开发了一种用 n 维 Hopfield 网络做联想存储器的结构。在这个网络中, 权系数的赋值规则为存储向量的外积存储规则, 其原理如下:

设有 m 个样本存储向量 x_1, x_2, \dots, x_m

$$\begin{aligned} x_1 &= \{x_{11}, x_{21}, \dots, x_{m1}\} \\ x_2 &= \{x_{12}, x_{22}, \dots, x_{m2}\} \\ &\vdots \\ x_m &= \{x_{1m}, x_{2m}, \dots, x_{mm}\} \end{aligned}$$

把这 m 个样本向量存储入 Hopfield 网络中, 则在网络中第 i, j 两个节点之间权系数的值为:

$$w_{ij} = \begin{cases} \sum_k x_{ik} \cdot x_{jk}, & i \neq j \\ 0, & i = j \end{cases} \quad (16-21)$$

其中, k 为样本向量 x_k 的下标, $k=1, 2, \dots, m$; i, j 分别是样本向量 x_k 的第 i, j 分量 x_i, x_j 的下标, $i, j=1, 2, \dots, n$ 。

对联想存储器的联想检索过程如下:

给定一个向量 X , 进行联想检索求取在网络中的存储内容。这时, 把向量 $X = \{x_1, x_2, \dots, x_n\}$ 的各个分量 x_1, x_2, \dots, x_n 赋给相对应的节点 ($j=1, 2, \dots, n$), 节点有相应的初始状态 $y_j(0)$, 则有

$$y_j(0) = x_j, \quad j=1, 2, \dots, n$$

接着, 在 Hopfield 网络中按动态系统原则进行计算, 得



$$Y_j(t+1) = f\left[\sum w_{ij}y_j(0) - \theta_j\right], \quad i, j = 1, 2, \dots, n \quad (16-22)$$

其中 f 是非线性函数，可取阶跃函数。

状态不断变化会稳定下来，最终的状态是和给定向量 X 最接近的样本向量。所以，Hopfield 网络的最终输出也就是给定向量联想检索结果。这个过程说明，即使给定向量并不完全或部分不正确，也能找到正确的结果。在本质上，它也有滤波功能。

16.2 连续型 Hopfield 网络

1984 年 Hopfield 把 DHNN 进一步发展成连续型 Hopfield 网络，缩写为 CHNN 网。CHNN 的基本结构与 DHNN 相似，但 CHNN 中所有神经元都同步工作，各输入-输出量均是随时间连续变化的模拟量，这就是使得 CHNN 比 DHNN 在信息处理的并行性、实时性等方面更接近于实际生物神经网络的工作机理。

CHNN 可以用常系数微分方程来描述，但用模拟电子线路来描述则更为形象直观，易于理解也便于实现。

1. 网络结构与数学模型

如果定义网络中第 i 个节点的输入 u_i ，输出为 v_i ，那么输入、输出的关系为：

$$v_i = f(u_i) = f\left(\sum_{j=1}^n w_{ji}v_j - \theta_i\right) \quad (16-23)$$

其中， n 为网络的节点数，状态转移函数为 Sigmoid 型函数，一般常用：

$$f(x) = 1 / (1 + e^{-\lambda x}) \text{ 或者 } f(x) = \text{th}(\lambda x)$$

网络工作方式分为异步、同步和连续更新三种方式，与离散型 Hopfield 网络相比，多了一种连续更新方式，即网络中所有节点的输出都随时间连续变化。

图 16-6 为利用运算放大器实现的神经元效果图。

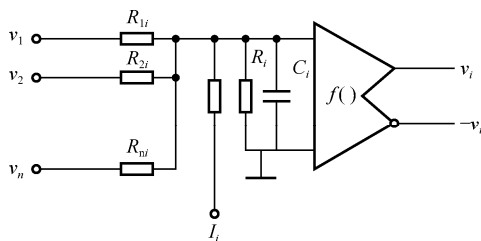


图 16-6 利用运算放大器实现的神经元

连续 Hopfield 神经网络模型如图 16-7 所示。

对图 16-6 的神经元，根据克希荷夫定律可写出下列微分方程：

$$C_i \frac{du_i}{dt} = \sum_{j=1}^n T_{ij}v_j - \frac{u_i}{R_i} + I_i \quad (16-24)$$

$$v_i = f(u_i)$$

不难看出，图 16-7 网络的数学模型可以用 n 个式 (16-24) 所表示的非线性方程组来描述。

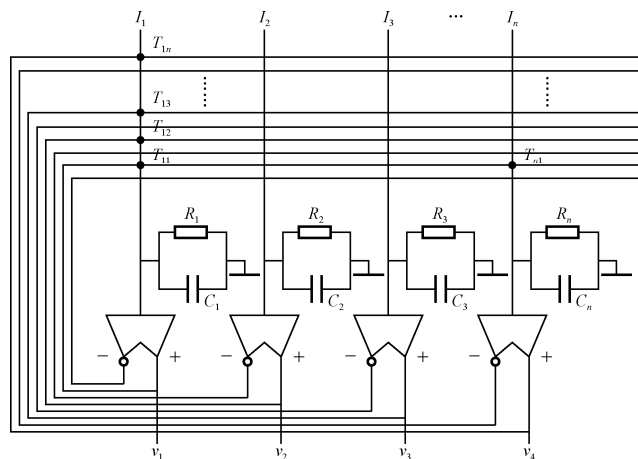


图 16-7 连续 Hopfield 神经网络模型

2. 网络稳定性

定义 CHNN 的能量函数为:

$$E = -\frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n w_{ij} v_i v_j - \sum_{j=1}^n v_j I_j + \sum_{j=1}^n \frac{1}{R_j} \int_0^{v_j} f^{-1}(v) dv \quad (16-25)$$

写成向量式为:

$$E = -\frac{1}{2} V^T W V - I^T V + \sum_{j=1}^n \frac{1}{R_j} \int_0^{v_j} f^{-1}(v) dv \quad (16-26)$$

式中, f^{-1} 为神经元转移函数的反函数。对于式 (16-25) 所定义的能量函数, 存在以下定理。

定理: 若神经元的转换函数 f 存在反函数 f^{-1} , 且 f^{-1} 是单调连续递增的, 同时网络权值对称, 即 $w_{ij} = w_{ji}$, 则由任意初态开始, CHNN 网络的能量函数总是单调递减的, 即 $\frac{dE}{dt} \leq 0$,

当且仅当 $\frac{du_j}{dt} = 0$ 时, 有 $\frac{dE}{dt} = 0$, 因而网络最终能够达到稳态。

证明: 将能量函数对时间求导, 得

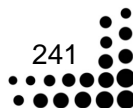
$$\frac{dE}{dt} = \sum_{j=1}^n \frac{\partial E}{\partial v_j} \frac{dv_j}{dt} \quad (16-27)$$

由式 (16-25) 和 $u_j = f^{-1}(v_j)$ 及网络的对称性, 对某神经元 j 有:

$$\frac{\partial E}{\partial v_j} = -\frac{1}{2} \sum_{i=1}^n w_{ij} v_i - I + \frac{u_j}{R_j} \quad (16-28)$$

将式 (16-28) 代入式 (16-27), 可整理得出下式:

$$\frac{dE}{dt} = \sum_{j=1}^n \frac{du_j}{dt} c_j \frac{dv_j}{dt} = -\sum_{j=1}^n c_j \frac{du_j}{dv_j} \left(\frac{dv_j}{dt} \right)^2$$





$$= \sum_{j=1}^n c_j f^{-1}(v_j) \left(\frac{dv_j}{dt} \right)^2$$

可以看出, 上式中 $c_j > 0$, 单调递增函数 $f^{-1}(v_j) > 0$, 故有

$$\frac{dE}{dt} \leq 0 \quad (16-29)$$

只有对于所有 j 均满足 $\frac{dv_j}{dt} = 0$ 时, 才有 $\frac{dE}{dt} = 0$ 。

如果运算放大器接近理想运放, 式 (16-25) 中的积分项可以忽略不计, 网络的能量函数可写为:

$$E = -\frac{1}{2} \sum_{j=1}^n \sum_{i=1}^n w_{ij} v_i v_j - \sum_{j=1}^n v_j I_j \quad (16-30)$$

Hopfield 网络用于联想记忆时, 正是利用了这些局部极小点来记忆样本, 网络的存储容量越大, 说明网络的局部极小点越多。然而在优化问题中, 局部极小点越多, 网络就越不容易达到最优解而只能达到较优解。

为保证网络的稳定性, 要求网络的结构必须对称, 否则运行中可能出现极限环或混沌状态。

3. 连续型 Hopfield 网络特点

连续型 Hopfield 网络具有如下几个特点:

- (1) 良好的收敛性;
- (2) 有限个平衡点;
- (3) 如果平衡点是稳定的, 那么它也一定是渐近稳定的;
- (4) 渐进稳定平衡点为其能量函数的局部极小点;
- (5) 能将任意一组希望存储的正交化向量综合为网络的渐近平衡点;
- (6) 网络的存储信息表现为神经元之间互连的分布式动态存储;
- (7) 网络以大规模、非线性、连续时间并行方式处理信息, 其计算时间就是网络趋于平衡点的时间。

16.3 Hopfield 神经网络的应用

16.3.1 Hopfield 神经网络函数

MATLAB 神经网络工具箱中包含了许多用于 Hopfield 网络分析与设计的函数, 下面介绍几个重要的函数。

1. newhop 函数

该函数用于创建一个离散型 Hopfield 神经网络, 其调用格式为:



```
net=newhop(T)
```

其中,

Net 为生成的神经网络, 具有在 T 中的向量上稳定的点;

T 为具有 Q 个目标向量的 $R \times Q$ 矩阵 (元素必须为 -1 或 1)。

Hopfield 神经网络经常被应用于模式的联想记忆中。Hopfield 神经网络仅有一层, 其激活函数用 `satlins()` 函数, 层中的神经元有来自它自身的连接权和阈值。

【例 16-2】 含有两个神经元的 Hopfield 网络的设计示例。

网络所要存储的目标平衡点为一个列向量 T :

```
T=[1 -1; -1 1];
```

在二维平面内绘制出这两个稳定平衡点, 如图 16-8 所示。原则上所设计的 Hopfield 网络的所有可能状态都应包含在图示所在的区间内, 所以称为 Hopfield 网络状态空间。

```
>> T=[1 -1; -1 1]
plot(T(1,:),T(2:),'rp');
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 网络状态空间');
xlabel('a(1)');ylabel('a(2)');
```

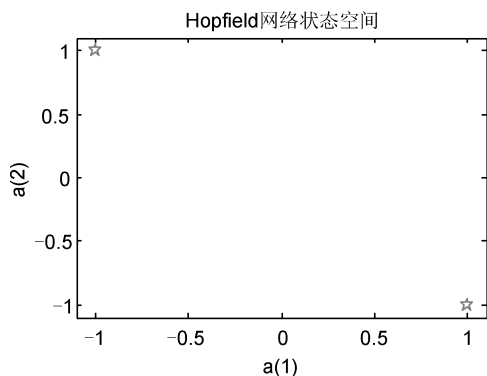


图 16-8 Hopfield 网络状态空间

应用 `newhop` 函数设计一个 Hopfield 网络, 满足期望值 T 的要求。

```
>> net=newhop(T);
```

应用期望值作为初始输入, 并应用 `sim` 函数进行检测。

```
>> [Y,Pf,Af]=sim(net,2,[],T);
>> T
T =
     1     -1
    -1     1
```

可见, 网络输出能够稳定到期望值。

再随机选取初始点, 然后应用 `sim` 函数进行仿真, 其结果应该稳定到两个平衡点之一。



```
>> a={rands(2,1)};
[y,Pf,Af]=sim(net,{1 20},{},a);
```

下面绘制出初始点向平衡点的逼近过程。

```
>> record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
hold on;
plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:))
```

运行程序，效果如图 16-9 所示。

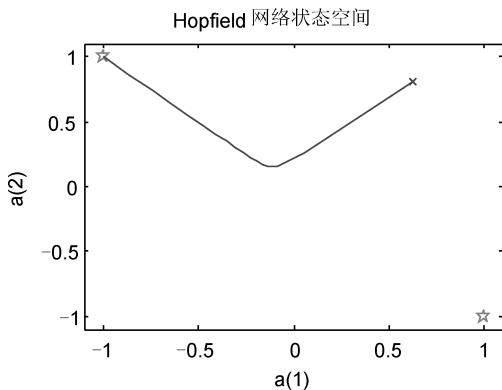


图 16-9 随机初始点测试

可见，随机选取的初始点最后稳定在左上角的平衡点。

下面再随机选取 25 个初始点进行测试。

```
>> color='rgbmy';
for i=1:25
    a={rands(2,1)};
    [y,Pf,Af]=sim(net,{1 20},{},a);
    record=[cell2mat(a) cell2mat(y)];
    start=cell2mat(a);
    plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:),color(mod(i,5)+1));
end
```

运行程序，效果如图 16-10 所示。

从图 16-10 中可以看出，所有的点最后都收敛于两个平衡点，与左上角接近的点收敛于左上角，与右下角接近的点收敛于右下角。由此可见，Hopfield 神经网络具有联想记忆功能。

2. satlins 函数

该函数为对称饱和线性传递函数，其调用格式为：

```
A = satlins(N)
```

其中，

A 为输出向量矩阵；

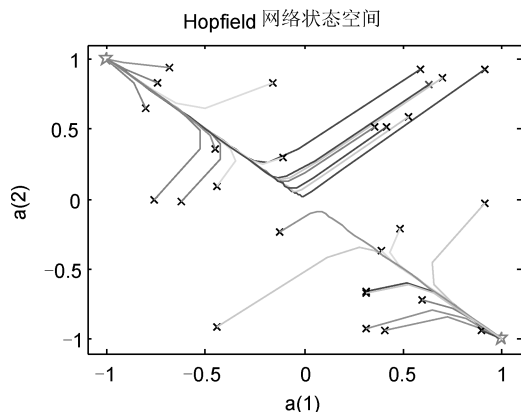


图 16-10 25 个随机初始点测试

N 为由网络的输入向量组成的 $S \times Q$ 矩阵, 返回的矩阵 A 与 N 的维数大小一致, A 的元素取值位于区间 $[0, 1]$ 内。当 N 中的元素介于 -1 和 1 之间时, 其输出等于输入; 当输入值小于 -1 时返回 -1 ; 当输入值大于 1 时返回 1 。

16.3.2 Hopfield 神经网络的应用

下面从几个方面来介绍 Hopfield 神经网络在实际领域中的应用。

1. 在数字识别中的应用

在日常生活中, 经常会遇到带噪声字符的识别问题, 如交通系统中的汽车牌照, 由于汽车在使用过程中, 要经受自然环境的风吹雨打, 造成字体模糊不清, 难以辨认。如何从这些残缺不全的字符中提取完整的信息是字符识别的关键问题。作为字符识别的组成部分之一, 数字识别在邮政、交通及商业票据管理方面存在极高的应用。

在此利用离散型 Hopfield 神经网络具有联想记忆的功能对数字进行识别, 可取得令人满意的效果, 并且计算的收敛速度很快。

【例 16-3】 设计一个 Hopfield 网络, 使其具有联想记忆功能, 能正确识别阿拉伯数字, 当数字被噪声污染后仍可以正确地识别。

假设网络由 10 个初始稳态值 $0 \sim 9$ 构成, 即可以记忆 10 种数字。每个稳态由 10×10 的矩阵构成, 该矩阵用于模拟阿拉伯数字点阵。所谓数字点阵, 就是将数字划分成很多小方块, 每一个小方块都对应一部分数字。这里将数字划分成一个 10×10 方阵, 其中, 有数字的方块用 1 表示, 空白处用 -1 表示, 如图 16-11 所示。

设计 Hopfield 网络需要经过以下几个步骤, 如图 16-12 所示。

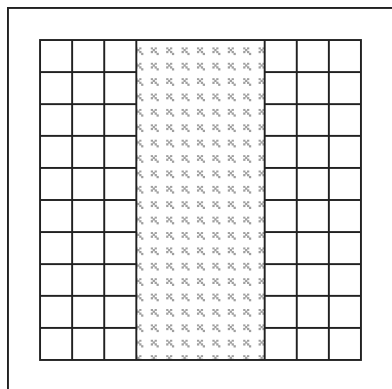


图 16-11 数字 1 的数字点阵图

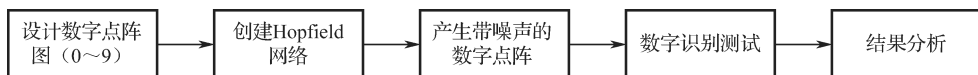


图 16-12 Hopfield 网络设计流程图

首先, 如图 16-16 所示, 即可得到数字 1 和数字 2 的点阵:

```

>> one=[-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 -1 -1 -1 -1;...
-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 -1 -1 -1 -1;...
-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 -1 -1 -1 -1;...
-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 -1 -1 -1 -1;...
-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 -1 -1 -1 -1];
two=[-1 1 1 1 1 1 1 1 1 -1;-1 1 1 1 1 1 1 1 1 -1;...
-1 -1 -1 -1 -1 -1 -1 -1 1 1;-1 -1 -1 -1 -1 -1 -1 -1 1 1;...
-1 1 1 1 1 1 1 1 1 -1;-1 1 1 1 1 1 1 1 1 -1;...
-1 1 1 -1 -1 -1 -1 -1 -1 -1;-1 1 1 -1 -1 -1 -1 -1 -1 -1;...
-1 1 1 1 1 1 1 1 1 -1;-1 1 1 1 1 1 1 1 1 -1];
  
```

其他数字的点阵依此类推。

利用这两个数字点阵构成训练样本 T:

```
>> T=[one;two];
```

利用 newhop 函数创建一个离散型 Hopfield 神经网络:

```
>> net=newhop(T);
```

带噪声的数字点阵, 即点阵的某些位置的值发生了变化。模拟产生带噪声的数字矩阵方法有很多种, 在此只介绍比较常用的两种, 分别为固定噪声产生法和随机噪声产生法。

1) 固定噪声产生法

固定噪声产生法又称人工产生法, 指的是用人工修改的方法改变数字点阵某些位置的值, 从而模拟产生带噪声的数字点阵。如数字 1 与数字 2 的点阵经过修改后的带噪声数字点阵变为:

```

>> noisy_one=[-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 -1 1 -1 1 -1 -1 -1 -1 -1;...
-1 -1 1 -1 1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 1 -1 -1 -1;...
-1 -1 -1 -1 -1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 1 -1 1 -1;...
-1 -1 -1 -1 1 1 -1 -1 -1 -1;-1 1 -1 -1 1 1 -1 -1 -1 -1;...
-1 -1 -1 -1 -1 1 -1 -1 -1 -1;-1 -1 -1 -1 1 1 1 -1 1 1];
noisy_two=[-1 1 1 1 -1 1 1 -1 1 -1;-1 1 1 1 1 1 1 1 1 -1;...
-1 -1 1 -1 1 -1 -1 1 1 -1;-1 1 1 1 1 1 1 1 1 -1;...
-1 1 1 1 1 1 1 1 1 -1;-1 1 1 1 1 1 1 1 1 -1;...
-1 1 1 -1 -1 -1 -1 -1 -1 -1;-1 1 1 -1 -1 -1 -1 -1 -1 -1;...
-1 1 1 1 -1 1 1 1 1 -1;-1 1 1 -1 1 1 1 1 1 -1];
  
```

如果希望产生不同的带噪声的数字矩阵, 需要人工做多次的修改, 这无疑是比较麻烦的。相比较而言, 随机噪声产生法可以方便地产生各种类型的带噪声的数字矩阵。



2) 随机噪声产生法

随机噪声产生法利用产生随机数的方法来确定需要修改的点阵位置,进而对数字点阵进行修改。由于数字点阵中的值只有 1 和 -1,所以此处的修改就是将“1”换成“-1”,将“-1”换成“1”。带噪声的数字 1 和 2 的数字点阵产生程序如下:

```
%随机噪声
noisy_one=one;
noisy_two=two;
for i=1:100
    a=rand;
    if a<0.15
        noisy_one(i)=-one(i);
        noisy_two(i)=-two(i);
    end
end
```

将带噪声的数字点阵输入已创建好的 Hopfield 网络,便可对带噪声的数字点阵进行识别。实现代码为:

```
>> %仿真测试
noisy_one2={noisy_one}';
identify_one=sim(net,{10,10},{},noisy_one2);
identify_one{10}' %获取复原后数字 1 点阵
noisy_two2={noisy_two}';
identify_two=sim(net,{10,10},{},noisy_two2);
identify_two{10}' %获取复原后数字 2 点阵
```

以图形的形式将点阵数字绘制出,程序代码为:

```
%绘图
subplot(3,2,1);one=imresize(one,20);
imshow(one);title('原始数据 1');
subplot(3,2,1);two=imresize(two,20);
imshow(two);title('原始数据 2');
subplot(3,2,3);noisy_one=imresize(noisy_one,20);
imshow(noisy_one);title('带噪声数字 1');
subplot(3,2,4);noisy_two=imresize(noisy_two,20);
imshow(noisy_two);title('带噪声数字 2');
subplot(3,2,5);one2=imresize(identify_one{10}',20);
imshow(one2);title('复原数字 1');
subplot(3,2,6);two2=imresize(identify_two{10}',20);
imshow(two2);title('复原数字 2');
```

运行程序,得到如图 16-13 所示效果。

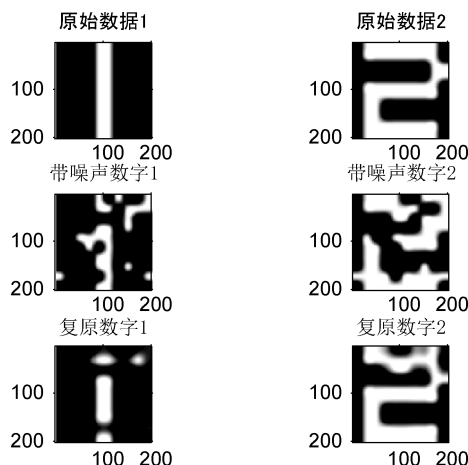


图 16-13 数字识别效果图

2. 高校科研能力评价

科研能力是高校的核心能力，其高低已成为衡量一所高校综合实力的重要指标。科研能力的高低不仅影响高校自身的发展，对高校所在地区的经济发展也有很大的影响。如何准确地评价高校的科研能力已成为摆在政府、企业和高校面前的一个重要问题。影响科研能力的因素诸多，且互相交错、互相渗透及相互影响，无法用确定的数学模型进行描述。目前，高校科研能力评价方法很多，但普遍存在工作烦琐、时间滞后等缺点，且人为主观因素对评价结果有很大的影响。如何快速、准确地对众多高校的科研能力进行客观、公正的评价，离散 Hopfield 神经网络可解决这个问题。

影响高校科研能力的因素很多，本例仅以较为重要的 11 个影响因素作为评价指标：科研队伍 (X_1)、科研基地 (X_2)、科技学识及其相应的载体 (图书情报资源) (X_3)、科研经费 (X_4)、科研管理 (X_5)、信息接收加工能力 (X_6)、学识积累与技术储备能力 (X_7)、科研技术创新能力 (X_8)、知识释放能力 (X_9)、自适应调节能力 (X_{10})、科学决策能力 (X_{11})。

高校科研能力一般分为 5 个等级：很强 (I)、较强 (II)、一般 (III)、较差 (IV) 及很差 (V)。

【例 16-4】 某机构对 20 所高校的科研能力进行了调研和评价，试根据调研结果中较为重要的 11 个评价指标数据，并结合离散 Hopfield 神经网络的联想记忆能力，建立离散 Hopfield 高校科研能力评价模型。

实现本实例的步骤主要包括如下 5 个步骤，如图 16-14 所示。

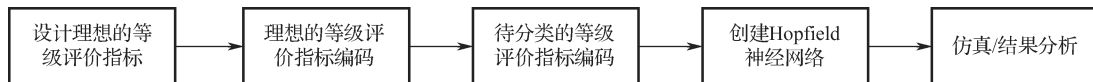


图 16-14 模型建立流程图

所研究的 20 所高校的科研能力等级与 11 个评价指标间的关系如表 16-1 所示。



表 16-1 20 所高校的科研能力等级及对应的评价指标

序号 \ 指标	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	等级
1	98	92	86	95	90	7	93	96	92	95	94	I
2	92	96	94	88	95	91	89	97	93	90	99	I
3	73	87	82	65	89	74	86	80	94	81	82	II
4	78	71	76	91	82	89	80	78	63	76	84	II
5	87	96	93	97	92	95	90	88	96	98	94	I
6	68	72	64	66	69	61	65	70	75	63	67	III
7	61	64	62	57	67	68	72	64	63	69	62	III
8	38	43	51	62	48	57	53	46	49	50	54	IV
9	53	46	47	58	55	36	39	48	52	58	47	IV
10	94	97	91	96	87	93	98	92	86	94	95	I
11	24	37	45	31	18	29	33	13	22	38	30	V
12	84	80	71	78	73	83	74	67	82	88	75	II
13	44	58	55	45	62	54	46	59	55	45	43	IV
14	35	23	16	27	38	24	29	28	38	21	26	V
15	16	44	32	38	26	35	20	37	34	33	39	V
16	65	67	68	62	61	58	63	69	64	62	66	III
17	58	65	62	67	71	69	64	65	70	74	65	III
18	73	84	95	78	84	86	76	83	89	75	87	II
19	33	28	35	20	26	44	38	26	30	44	21	V
20	94	89	96	94	91	99	95	87	93	88	88	I

将各等级的样本对应的各评价指标的平均值作为各个等级的理想评价指标，即作为 Hopfield 神经网络的平衡点，如表 16-2 所列。

表 16-2 5 个等级理想评价指标

等级 \ 指标	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}
I	93	94	92	94	91	95	93	92	92	93	94
II	77	78	81	78	82	83	79	77	82	80	82
III	63	67	64	63	67	64	66	67	68	67	65
IV	45	49	51	55	55	49	46	51	52	51	48
V	27	33	32	29	27	33	30	26	31	34	29

由于离散型 Hopfield 神经网络神经元的状态只有 1 和-1 两种情况，所以将评价指标映射



为神经元的状态时，需要将其进行编码。编码规则为：当大于或等于某个等级的指标值时，对应的神经元状态设为“1”，否则设为“-1”。理想的 5 个等级评价编码如图 16-15 所示。其中●表示神经元状态“1”，即大于或等于 1 对应等级的理想评价指标值，反之则用○表示。

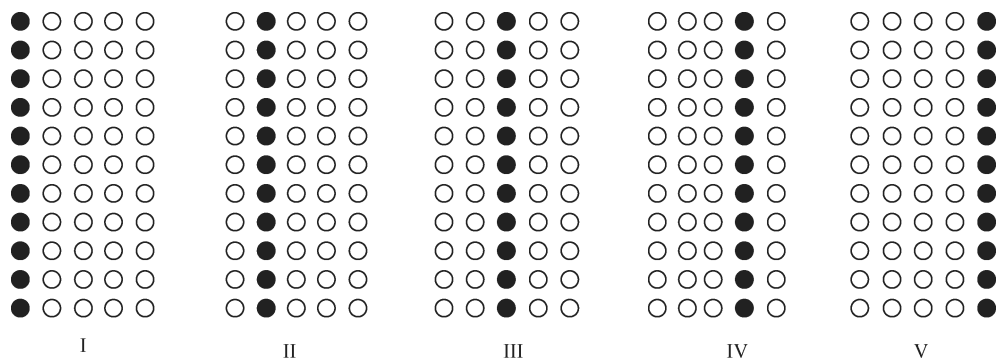


图 16-15 理想的 5 个等级评价指标编码

5 所待分类的高校等级评价指标如表 16-3 所列，根据上述的编码规则得到对应的编码，如图 16-16 所示。

表 16-3 5 所分类的高校高级评价指标

指标 序号	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}
1	96	92	85	89	93	87	94	76	98	94	97
2	70	88	75	82	96	79	89	80	84	85	83
3	60	75	68	67	57	74	76	83	69	75	64
4	55	59	41	81	58	73	57	48	56	43	55
5	20	38	42	25	24	37	40	36	21	46	35

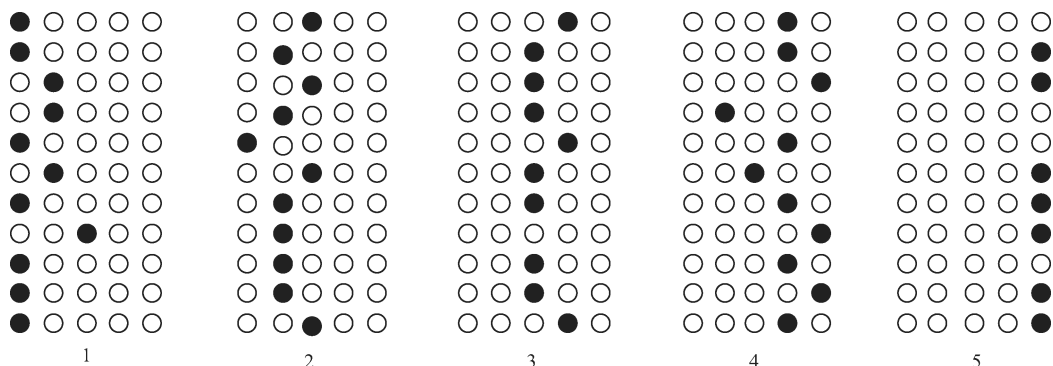


图 16-16 5 所待分类的高校等级评价指标编码

首先清除工作空间，实现代码为：

```
>> clear all;
```



理想的 5 个等级评价指标编码为 5 个 11×5 的矩阵，每个矩阵中的元素只包含“1”和“-1”两种取值。数据编码情况如下：

```
>> class_1=[1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1;...
            1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1;...
            1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1];
class_2=[-1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1;...
          -1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1;...
          -1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1];
class_3=[-1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1;...
          -1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1;...
          -1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1];
class_4=[-1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1;...
          -1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1;...
          -1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1];
class_5=[-1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;...
          -1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;...
          -1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1];
```

待分类的 5 所高校等级评价指标如下所示：

```
>> sim_1=[1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1;...
          1 -1 -1 -1 -1;-1 1 1 -1 -1;1 -1 -1 -1 -1;-1 1 1 -1 -1;...
          1 -1 -1 -1 -1;1 -1 -1 -1 -1;1 -1 -1 -1 -1];
sim_2=[-1 -1 1 1 -1;-1 1 1 -1 -1;-1 -1 1 1 -1;-1 1 1 -1 -1;...
        1 -1 -1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1;...
        -1 1 1 -1 -1;-1 1 1 -1 -1;-1 1 1 -1 -1];
sim_3=[-1 -1 -1 1 1;-1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1;...
        -1 -1 -1 1 1;-1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1;...
        -1 -1 1 1 -1;-1 -1 1 1 -1;-1 -1 1 1 -1];
sim_4=[-1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1;-1 1 1 -1 -1;...
        -1 -1 -1 1 1;-1 1 1 -1 -1;-1 -1 -1 1 1;-1 -1 -1 1 1;...
        -1 -1 -1 1 1;-1 -1 -1 1 1;-1 -1 -1 1 1];
sim_5=[-1 -1 -1 -1 -1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;...
        -1 -1 -1 -1 -1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;-1 -1 -1 -1 1;...
        -1 -1 -1 -1 -1;-1 -1 -1 -1 1;-1 -1 -1 -1 1];
```

将理想的 5 个等级评价指标的编码作为 Hopfield 神经网络的平衡点，代码为：

```
T=[class_1 class_2 class_3 class_4 class_5];
```

创建离散型的 Hopfield 神经网络，代码为：

```
net=newhop(T);
```

将待分类的 5 所高校等级评价指标的编码输入创建好的离散型 Hopfield 神经网络，进行仿真，实现代码为：



```
>> %网络仿真
A={sim_1 sim_2 sim_3 sim_4 sim_5}};
Y=sim(net,{25 20},{},A);
Y1=Y{20}(:,1:5)
Y2=Y{20}(:,6:10)
Y3=Y{20}(:,11:15)
Y4=Y{20}(:,16:20)
Y5=Y{20}(:,21:25)
```

以图形的形式显示结果，代码为：

```
>> %结果显示
result={T;A{1};Y{20}};
for p=1:3
    for k=1:5
        subplot(3,5,(p-1)*5+k);
        temp=result{p}(:,(k-1)*5+1:k*5);
        [m,n]=size(temp);
        for i=1:m
            for j=1:n
                if temp(i,j)>0
                    plot(j,m-i,'ko','MarkerFacecolor','k');
                else
                    plot(j,m-i,'ko');
                end
            end
            hold on;
        end
    end
    axis([0 6 0 12]);
    axis off;
    if p==1
        title(['class' num2str(k)])
    elseif p==2
        title(['presim' num2str(k)])
    else
        title(['sim' num2str(k)])
    end
end
end
```

运行程序，得到仿真测试结果如下，效果如图 16-17 所示。

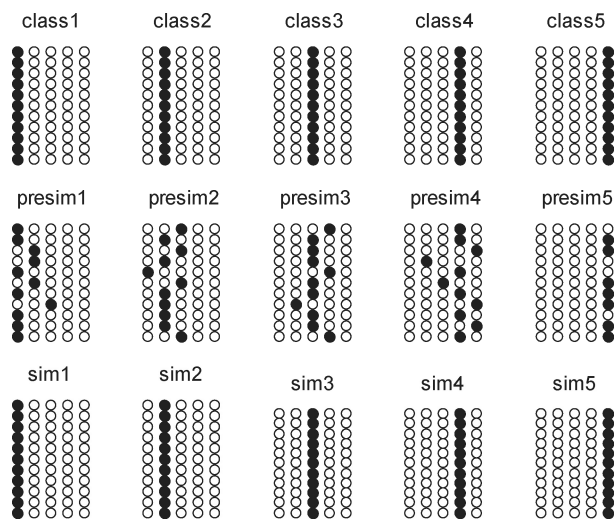


图 16-17 待分类 5 所高校等级评价指标编码仿真结果

Y1 =

1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1
1	-1	-1	-1	-1

Y2 =

-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1
-1	1	-1	-1	-1

Y3 =

-1	-1	1	-1	-1
-1	-1	1	-1	-1



```

-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1
-1    -1     1    -1    -1

```

Y4 =

```

-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1
-1    -1    -1     1    -1

```

Y5 =

```

-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1
-1    -1    -1    -1     1

```

图 16-17 第一行与图 16-15 相对应,表示 5 个理想的等级评价指标编码;第二行与图 16-16 相对应,表示 5 所待分类的高校等级评价指标编码;第三行为设计的 Hopfield 神经网络分类的结果。从图可清晰地看出,设计的 Hopfield 网络可有效地进行分类,从而可对高校的科研能力进行客观公正的评价。

值得注意的是,离散型 Hopfield 神经网络并非适用于任何场合。当某所高校的优势与劣势并存且相当明显(一些影响因素得分很高,一些影响因素得分很低)时, Hopfield 神经网络将得不到确切的分类。如某所高校经过上述的编码规则编码后,得到的等级评价指标编码为:



```
>> sim_n=[1 -1 -1 -1 -1;-1 -1 -1 1 -1;-1 1 -1 -1 -1;-1 1 -1 -1 -1;...
1 -1 -1 -1 -1;-1 -1 1 -1 -1;-1 -1 -1 1 -1;-1 -1 -1 -1 1;...
-1 1 -1 -1 -1;-1 -1 -1 1 -1;-1 -1 1 -1 -1];
```

利用前面创建好的 Hopfield 网络进行仿真，仿真步数 TS 设置为 100，实现代码为：

```
y=sim(net,{5,100},{},{sim_n});
a=y(100)
```

得到仿真结果如下：

```
a=
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
```

如图 16-18 所示，从仿真结果可看出，它的不属于 5 种典型等级类别，即意味着所设计的 Hopfield 神经网络寻找不到与之最为接近的平衡点，因此无法将其正确分类。

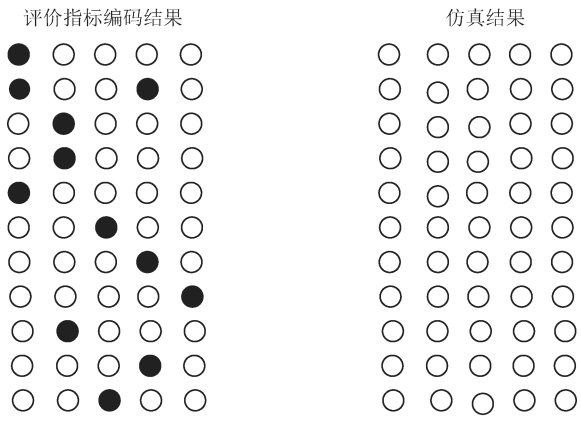


图 16-18 待分类的高校评价指标编码与仿真结果

3. Hopfield 神经网络应用于联想记忆的 MATLAB 程序

【例 16.5】 Hopfield 神经网络用于联想记忆。假设希望构造一个联想记忆模型，这个模型能够识别出如图 16-19 所示的 4 个数字。

图 16-19 所示二值化数字图像为二维图像，若以 0 表示数字笔画画过的小方块，以 1 表示数字笔画未画过的小方块，则 Hopfield 网络需要 56 个神经元表示各方块的状态，同时要求



目标向量是一维的，用一维向量表示。

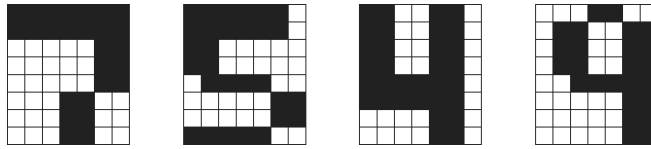


图 16-19 待识别数字的 8×7 二值化图像

例如，数字“7”的目标向量如下：

```
T7=[0000000
     0000000
     1111100
     1111100
     1111100
     1110011
     1110011
     1110011];
```

设计 Hopfield 神经网络的 MATLAB 程序如下：

```
T7=[0000000
     0000000
     1111100
     1111100
     1111100
     1110011
     1110011
     1110011];
>> clear all;
T7=[0000000
     0000000
     1111100
     1111100
     1111100
     1110011
     1110011
     1110011];
T5=[0000001
     0000001
     0011111
     0011111
     1000011
     1111100
     1111100]
```




```

    0000011];
T4=[0011001
    0011001
    0011001
    0011001
    0000001
    0000001
    1111001
    1111001];
T9=[1110011
    1001100
    1001100
    1001100
    1100000
    1111100
    1111100
    1111100];
%形成总的目标向量
T=[T7 T5 T4 T9];
%设计 Hopfield 网络
net=newhop(T);

```

以加噪并且产生畸变的数字“7”作为测试对象，对所设计的 Hopfield 神经网络进行仿真。

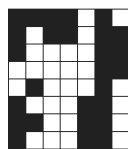
```

T7=[0000000
    0000000
    1111100
    1111100
    1111100
    1110011
    1110011
    1110011];
subplot(121);
%绘制测试样本二值化图像
figt(T7);
%网络仿真
y=sim(net,1,[],T7);
%二值化
y=y>0.5;
subplot(122);
%输出仿真输出二值化图像
figt(y)

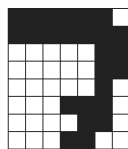
```

仿真结果如图 16-20 所示。从结果上可以看出，受到噪声影响并且严重畸变的数字“7”，

通过网络后的输出已非常接近网络存储的对应于“7”的样本模式。



(a) 测试样本



(b) 仿真结果

图 16-20 测试样本和仿真结果的二值化图像

源代码中的 `figt(t)` 是绘制测试样本二值化图像的自定义函数，其源代码如下：

```
%绘制测试样本二值化图像的自定义函数 figt(t)
function figt(t)
hold on
axis square
for j=1:8
    for i=1:7
        if t((j-1)*7+i)==0
            fill([i i+1 i+1 i],[9-j,9-j,10-j,10-j],'k')
        else
            fill([i i+1 i+1 i],[9-j,9-j,10-j,10-j],'w')
        end
    end
end
end
hold off
```

第 17 章 LVQ 网络算法分析与应用

学习向量量化 LVQ (Learning Vector Quantization) 神经网络属于前向有监督神经网络类型, 在模式识别和优化领域有着广泛的应用, 由芬兰学者 Teuvo Kohonen 提出。LVQ 神经网络由输入层、隐含层和输出层三层组成, 输入层与隐含层间为完全连接, 每个输出层神经元与隐含层神经元的不同组相连接。隐含层和输出层神经元之间的连接权值固定为 1。在网络训练过程中, 输入层和隐含层神经元间的权值被修改。当某个输入模式被送到网络时, 最接近输入模式的隐含神经元因获得激发而赢得竞争, 因而允许它产生一个“1”, 而其他隐含层神经元都被迫产生“0”。与包含获胜神经元的隐含层神经元组相连接的输出神经元也发出“1”, 而其他输出神经元均发出“0”。

17.1 LVQ 神经网络的结构

因为对于学习向量量化 (LVQ) 网络用户指定了目标分类结果, 所以网络可以通过监督学习完成对输入向量模式的准确分类。LVQ 神经网络模型如图 17-1 所示。

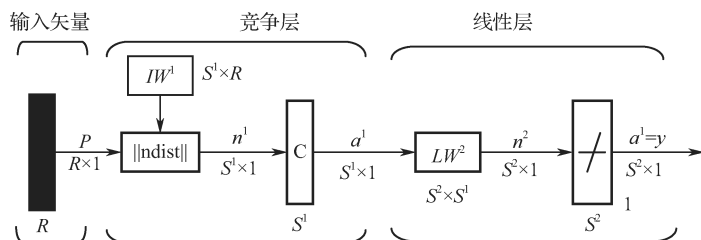


图 17-1 LVQ 神经网络模型

LVQ 神经网络有两个网络层, 即竞争层和线性层。竞争层对输入向量的学习分类与前面所阐述的竞争层一样, 我们把竞争层的分类称为子分类; 线性层根据用户的要求将竞争层的分类结果映射到目标分类结果中, 我们把线性层的分类称为目标分类。

竞争层和线性层的每一个神经元的输出都是对应一个分类 (子分类或目标分类) 结果, 所以竞争层通过学习可以得到 S^1 类子分类结果; 然后, 线性层将 S^1 类子分类结果再分成 S^2 类目标分类结果 (S^1 始终大于 S^2)。例如, 假设竞争层的第 1, 2, 3 个神经元对输入空间的子分类所对应的线性层的目标分类为第 2 类, 则竞争层的第 1, 2, 3 个神经元与线性层的第 2 个神经元的连接权将全部为 1, 而与其他线性层神经元的连接权全部为 0, 这样, 当竞争层的第 1, 2, 3 个神经元中的任意一个神经元在竞争中获胜时, 线性层的第 2 个神经元将输出 1。



17.2 LVQ 神经网络的学习算法

LVQ 神经网络算法是在有教师状态下对竞争层进行训练的一种学习算法,因此 LVQ 算法可以认为是把自组织特征映射算法改良成有教师学习的算法。LVQ 神经网络算法可分为 LVQ1 算法和 LVQ2 算法两种。

17.2.1 LVQ1 算法

向量量化是利用输入向量的固有结构进行数据压缩的技术,学习向量量化是在向量量化基础上能将输入向量分类的监督学习技术。Kohonen 把自组织特征映射算法改良成有教师学习算法,首先设计了 LVQ1 算法。LVQ1 的训练过程开始于随机地从“标定”训练集合选择一个输入向量以及该向量的正确类别。

LVQ1 算法的基本思想:计算距离输入向量最近的竞争层神经元,从而找到与之相连接的线性输出层神经元,如果输入向量的类别与线性输出层神经元所对应的类别一致,则对应的竞争层神经元权值沿着输入向量的方向移动;反之,如果两者的类别不一致,则对应的竞争层神经元权值沿着输入向量的反方向移动。基本的 LVQ1 算法步骤为:

(1) 初始化输入层与竞争层之间的权值 w_{ij} 及学习率 $\eta(\eta > 0)$ 。

(2) 将输入向量 $x = (x_1, x_2, \dots, x_R)^T$ 送入到输入层,并根据式(17-1)计算竞争层神经元与输入向量的距离:

$$d_i = \sqrt{\sum_{j=1}^R (x_j - w_{ij})^2}, i = 1, 2, \dots, S \quad (17-1)$$

式中, w_{ij} 为输入层的神经元 j 与竞争层的神经元 i 之间的权值。

(3) 选择与输入向量距离最小的竞争层神经元。如果 d_i 最小,则记与之连接的线性输出层神经元的类标签为 C_i 。

(4) 记输入向量对应的类标签为 C_x , 如果 $C_i = C_x$, 则根据式(17-2)调整权值, 否则, 根据式(17-3)进行权值更新:

$$w_{ij_new} = w_{ij_old} + \eta(x - w_{ij_old}) \quad (17-2)$$

$$w_{ij_new} = w_{ij_old} - \eta(x - w_{ij_old}) \quad (17-3)$$

17.2.2 LVQ2 算法

在 LVQ1 算法中,只有一个神经元可以获胜,即只有一个神经元的权值可以得到更新调整。为了提高分类的正确率, Kohonen 改进了 LVQ1, 被称为新版本 LVQ2。LVQ2 算法基于光滑的移动块决策边界逼近 Bayes 极限。LVQ2 版本接着被修改,产生 LVQ2.1 并且最终发展为 LVQ3。这些后来的 LVQ 版本共同具有的特点是引入了“次获胜”神经元,获胜神经元的



权值向量和“次获胜”神经元的权值向量都被更新，其计算步骤为：

(1) 利用 LVQ1 算法对所有输入模式进行学习。

(2) 将输入向量 $\mathbf{x} = (x_1, x_2, \dots, x_R)^T$ 送入输入层，并根据式 (17-1) 计算竞争层与输入向量的距离。

(3) 选择与输入向量距离最小的两个竞争层神经元 i, j 。

(4) 如果神经元 i 和神经元 j 满足以下两个条件：

① 神经元 i 和神经元 j 对应于不同的类别；

② 神经元 i 和神经元 j 与当前输入向量的距离 d_i 和 d_j 满足式 (17-4)：

$$\min \left(\frac{d_i}{d_j}, \frac{d_j}{d_i} \right) > \rho \quad (17-4)$$

式中， ρ 为输入向量可能落进的接近于两个向量中段平面的窗口宽度，一般取 2/3 左右。

则有，

① 如果神经元 i 对应的类别 C_i 与输入向量对应的类别 C_x 一致，即 $C_x = C_i$ ，则神经元 i 和神经元 j 的权值根据式 (17-3) 进行修正：

$$\begin{aligned} w_{i_new} &= w_{i_old} + \alpha(x - w_{i_old}) \\ w_{j_new} &= w_{j_old} - \alpha(x - w_{j_old}) \end{aligned} \quad (17-5)$$

② 如果神经元 j 对应的类别 C_j 与输入向量对应的类别 C_x 一致，则 $C_x = C_j$ ，则神经元 i 和神经元 j 的权值根据式 (17-6) 进行修正：

$$\begin{aligned} w_{i_new} &= w_{i_old} - \alpha(x - w_{i_old}) \\ w_{j_new} &= w_{j_old} + \alpha(x - w_{j_old}) \end{aligned} \quad (17-6)$$

(5) 如果神经元 i 和神经元 j 不满足 (4) 中的条件，即只更新距离输入向量最近的神经元权值，更新公式与 LVQ2 算法中 (1) 相同。

17.3 LVQ 神经网络的特点

竞争层神经网络可以自动学习对输入向量模式的分类，但是竞争层进行的分类只取决于输入向量之间的距离，当两个输入向量非常接近时，竞争层就可能将它们归为一类。在竞争层的设计中没有这样的机制，即严格地判断任意的两个输入向量是属于同一类还是属于不同类。面对于 LVQ 网络用户指定目标分类结果，网络可以通过监督学习完成对输入向量模式的准确分类。

与其他模式识别和映射方式相比，LVQ 神经网络的优点在于网络结构简单，只通过内部单元的相互作用就可以完成十分复杂的分类处理，也很容易将设计域中的各种繁杂分散的设计条件收敛到结论上来。而且它不需要对输入向量进行归一化、正交化处理，只需要直接计算输入向量与竞争层之间的距离，从而实现模式识别，因此简单易行。



17.4 LVQ 神经网络的 MATLAB 函数

在 MATLAB 神经网络工具箱提供了若干函数用于实现 LVQ 神经网络。

1. 创建函数 newlvq

该函数用于创建一个学习向量量化 LVQ 网络，其调用格式为：

```
net=newlvq  
net=newlvq(PR,S1,PC,LR,LF)
```

其中，

net=newlvq: 用于在对话框中创建一个 LVQ 网络；

PR: $R \times 2$ 的矩阵，指定了输入向量中元素的最大值和最小值；

S1: 竞争层神经元的数目；

PC: 分类的百分比；

LR: 学习速率，默认为 0.01；

LF: 学习函数，默认为 learnlv1。

【例 17-1】 根据给定的向量创建一个向量量化网络，并求其权值。

```
>> clear all;  
P=[-3 -2 -2 0 0 0 2 2 3; 0 1 -1 2 1 -1 -2 1 -1 0];  
net=newlvq(P,4,[.6 .4]);           %创建学习向量量化网络  
net.IW{1,1}  
net.LW{2,1}
```

运行程序，输出如下：

```
ans =  
      0      0  
      0      0  
      0      0  
      0      0  
  
ans =  
      1      1      0      0  
      0      0      1      1
```

2. 学习函数

在 LVQ 神经网络中有 LVQ1 算法及 LVQ2 算法，因此 MATLAB 提供了对应的两个学习函数。

(1) learnlv1 函数

该函数为 LVQ1 算法对应的权值学习函数，其调用格式为：

```
[dW,LS]=learnlv1(W,P,Z,N,A,T,E,gW,gA,D,L,LS)
```



其中,

dW 为权值 (或阈值) 变化矩阵。

LS 为当前学习状态 (可省略)。

W 为权值矩阵或者是阈值向量。

P 为输入向量或者是全为 1 的向量。

Z 为输入层的权值向量 (可省略)。

N 为网络的输入向量 (可省略)。

A 为网络的输出向量。

T 为目标输出向量 (可省略)。

E 为误差向量 (可省略)。

gW 为与性能相关的权值梯度矩阵 (可省略)。

gA 为与性能相关的输出梯度矩阵。

D 为神经元的距离矩阵。

LP 为学习参数, 默认值为 0.01。

LS 为初始学习状态。

【例 17-2】 对给出的随机矩阵创建学习向量量化网络, 并用 LVQ1 算法进行学习。

```
>> clear all;
p = rand(2,1);
w = rand(3,2);
a = compet(negdist(w,p));
gA = [-1;1; 1];
lp.lr = 0.5;
dW = learnlv1(w,p,[],[],a,[],[],gA,[],lp,[])
```

运行程序, 输出如下:

```
dW =
    -0.0024    -0.0768
         0         0
         0         0
```

(2) learnlv2 函数

该函数为 LVQ2 算法对应的权值学习函数, 其调用格式为:

```
[dW,LS] = learnlv2(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
```

其参数意义与 learnlv1 中的参数意义相同, 只是权值调整的方法不同。

3. 显示函数 plotvec

该函数用不同颜色绘制向量的函数, 其调用格式为:

```
plotvec(X,C,M)
```

其中,

X 为一个列向量矩阵。



C 为标记颜色坐标的行向量。

M 为指定绘图时向量的标记符号，默认值为“+”。

17.5 LVQ 神经网络的应用

下面介绍 LVQ 神经网络在模式分类中的应用。

【例 17-3】 假设有 10 个输入向量，构建一个学习向量量化网络，使这些输入向量分别指向 4 个子类中的一个。这样我们设定竞争层有 4 个神经元，这些子类又指向线性层 2 个神经元对应两个期望类别中的一个。

输入向量和期望输出类别如下：

```
>> clear all;  
P=[-3 -2 -2 0 0 0 2 2 3;0 1 -1 2 1 -1 -2 1 -1 0];  
Tc=[1 1 1 2 2 2 2 1 1 1];  
plotvec(P,Tc);
```

输入向量分布图如图 17-2 所示。

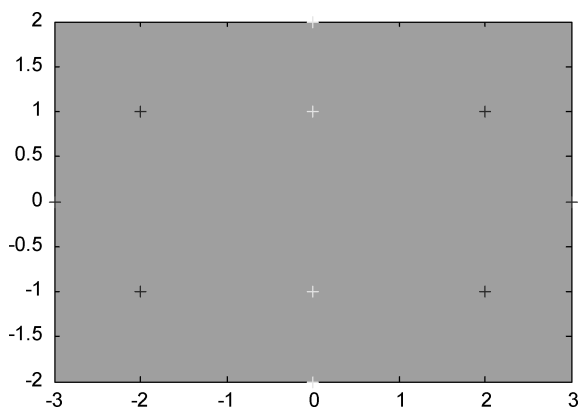


图 17-2 输入向量与期望向量散点图

分类的目的是，将输入量 p(1)、p(2)、p(3)、p(8)、p(9)、p(10)归类为输出 1，将输入量 p(4)、p(5)、p(6)、p(7)归类为输出 2，这个问题是非线性分类问题，因此不能应用感知器网络解决，而若应用 LVQ 网络就不难解决。

下面，将 Tc 矩阵转换为目标向量。

```
>> T=ind2vec(Tc);
```

从而得到稀疏矩阵 T，完整显示如下：

```
>> targets=full(T)
```

targets =

1	1	1	0	0	0	0	1	1	1
0	0	0	1	1	1	1	0	0	0



使用函数 `train` 训练网络，设置训练次数为 150。

```
>> net=newlvq(minmax(P),4,[0.6 0.4],0.1);
net.trainParam.epochs=150;
net.trainParam.show=Inf;
net=train(net,P,T);
```

竞争层权值为：

```
>> net.IW{1,1}
ans =
    1.1629   -0.0656
   -0.6320   -0.0190
         0   -0.2800
         0         0
```

此时权值移向期望类别，如图 17-3 所示。

```
>> cla;
plotvec(P,Tc);
hold on;
plotvec(net.IW{1}',vec2ind(net.LW{2}),'rp');
```

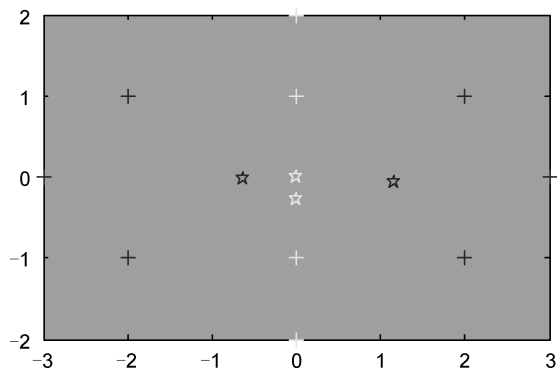


图 17-3 训练后网络竞争层权值

仿真网络可以验证这个权值确实对输入向量 P 进行了正确的分类。

```
>> Y=sim(net,P);
Yc=vec2ind(Y)
Yc =
     1     1     1     2     2     2     2     1     1     1
```

选择不同的输入，应用以上训练好的 LVQ 网络进行分类。

```
>> pk1=[0;0.5];
Y=sim(net,pk1);
Yc1=vec2ind(Y)
Yc1 =
```



```

2
>> pk2=[1;0];
Y=sim(net,pk2);
Yc2=vec2ind(Y)
Yc2 =
1

```

【例 17-4】 演示如何应用 LVQ 网络依据目标类别对输入向量进行分类。

首先给出二维输入向量 P ，以及其相应所属的类别 C ，并应用 `ind2vec` 函数将 C 转换为向量形式 T 。

```

>> clear all;
P=[-3 -2 -2 0 0 0 0 +2 +2 +3;0 +1 -1 +2 +1 -1 -2 +1 -1 0];
C=[1 1 1 2 2 2 2 1 1 1];
T=ind2vec(C);
i=1;
cla
for i=1:10
    if C(i)==1
        plot(P(1,i),P(2,i),'+')
        hold on
    else
        plot(P(1,i),P(2,i),'rp')
        hold on
    end
end
title('输入向量');
xlabel('P(1)'); ylabel('P(2)');

```

运行程序，输出数据点效果如图 17-4 所示。

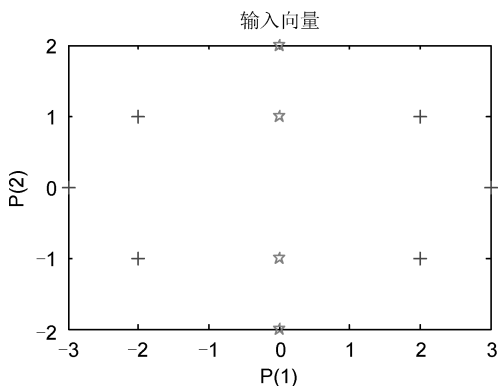


图 17-4 输出数据点效果

图 17-4 中，“+”标示的数据点代表第一类，“星号”标示的数据点代表第二类，要求设计一个 LVQ 网络对这些数据点集进行分类，网络的输出表示相应的分类结果。



应用 `newlvq` 函数构建一个 LVQ 网络，函数需要设定 4 个参数，第一个参数表示输入向量的区间。应用 `minmax` 函数求取，第二个参数表示隐层神经元的数目为 4 个，第三个参数为典型类别的百分比元素，第四个参数为学习速率。

```
net=newlvq(minmax(P),4,[.6 .4],0.1);
```

绘制初始网络竞争神经元的权值向量。

```
hold on
W1=net.iw{1};
plot(W1(1,1),W1(1,2),'*');
title('输入/权值向量');
xlabel('P(1),W(1)'); ylabel('P(2),W(2)');
```

运行程序，效果如图 17-5 所示。

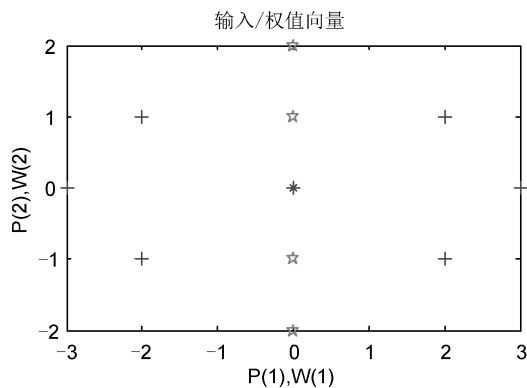


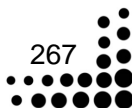
图 17-5 输入向量和初始网络权值

设置网络训练参数，应用 `train` 函数对前面所构建的网络加以训练。

```
net.trainParam.epochs=150;
net.trainParam.show=Inf;
net=train(net,P,T)
```

将输入向量和训练后的网络权值向量绘制在一张图上。

```
W1=net.IW{1};
W2=vec2ind(net.LW{2});
i=1;
cla
for i=1:10
    if C(i) == 1
        plot(P(1,i),P(2,i),'p')
        hold on
    else
        plot(P(1,i),P(2,i),'o')
        hold on
    end
end
```





```

        end
    end
    j=1;
    for j=1:4
        if W2(j)==1;
            plot(W1(j,1),W1(j,2),'p','markersize',15);
            hold on;
        else
            plot(W1(j,1),W1(j,2),'o','markersize',15);
            hold on;
        end
    end
    end
    title('输入/权值向量');
    xlabel('P(1),W(1)');
    ylabel('P(2),W(2)');

```

运行程序，效果如图 17-6 所示。

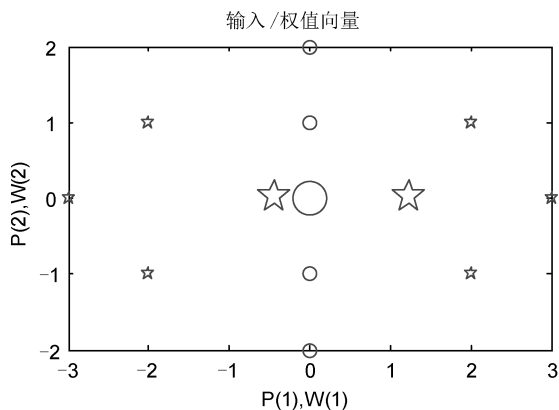


图 17-6 输入向量和训练后的权值

经过训练以后，竞争神经元的权值向量已经具有模式分类的功能，其中大“星号”表示第一类，大“o”表示第二类。

下面应用向量 p 通过 `sim` 函数仿真对网络进行测试，然后将输出向量转换为指针。

```

p=[0.2;1];
a=vec2ind(sim(net,p))
a =
    2

```

网络输出 $a = 2$ ，表示输入向量 p 归属于第 2 类。

第 18 章 自组织网络算法分析与实现



在生物神经系统中，存在着一种侧抑制现象，即一个神经细胞兴奋以后，会对周围其他神经细胞产生抑制作用。这种抑制作用会使神经细胞之间出现竞争，其结果是某些获胜，而另一些则失败。表现形式是获胜神经细胞兴奋，失败神经细胞抑制。

自组织竞争型神经网络就是模拟上述生物神经系统功能的人工神经网络。自组织竞争型神经网络是一种无教师监督学习，具有自组织功能的神经网络。网络通过自身的训练，能自动对输入模式进行分类。这一点与 Hopfield 网络的模拟人类功能十分相似，自组织竞争型神经网络的结构及其学习规则与其他神经网络相比有自己的特点。

在网络结构上，它一般是由输入层和竞争层构成的两层网络。两层之间各神经元实现双向连接，而且网络没有隐含层。有时竞争层各神经元之间还存在横向连接。

在学习算法上，它模拟生物神经元之间的兴奋、协调与抑制、竞争作用的信息处理的动力学原理来指导网络的学习与工作，而不像大多数神经网络那样是以网络的误差或能量函数作为算法的准则。

竞争型神经网络构成的基本思想是网络的竞争层各神经元竞争对输入模式响应的机会，最后仅有一个神经元成为竞争的胜者。这一获胜神经元则表示对输入模式的分类。

自组织竞争人工神经网络是基于上述生物结构和现象形成的。它能够对输入模式进行自组织训练和判断，并将其最终分为不同的类型。

与 BP 网络相比，这种自组织自适应的学习能力进一步拓宽了人工神经网络在模式识别、分类方面的应用，另一方面，竞争学习网络的核心——竞争层，又是许多种其他神经网络模型的重要组成部分。

常用自组织网络有：

- 自组织特征映射（Self-Organizing Feature Map, SOM）网络；
- 学习向量量化（Learning Vector Quantization, LVQ）网络；
- 自适应共振理论（Adaptive Resonance Theory, ART）模型；
- 对偶传播（Counter propagation, CP）网络。

自组织网络结构上属于层次型网络，有多种类型，其共同特点是都具有竞争层。最简单的网络结构具有一个输入层和一个竞争层，如图 18-1 所示。输入层负责接收外界信息并将输入模式向竞争层传递，起“观察”作用，竞争层负责对该模式进行“分析比较”，找出规律以正确归类，这种功能是通过下面要介绍的竞争机制实现的。

其中，

- 输入层：接收外界信息，将输入模式向竞争层传递，起“观察”作用。
- 竞争层：负责对输入模式进行“分析比较”，寻找规律并归类。

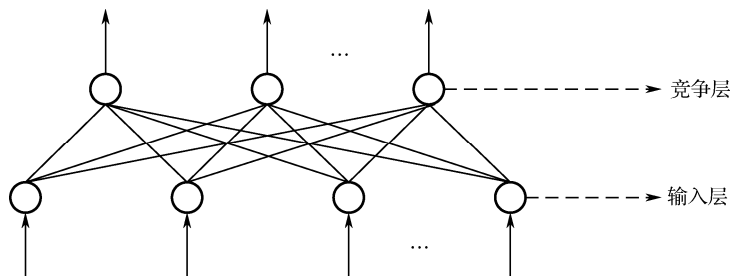


图 18-1 自组织网络的典型结构

18.1 竞争学习的概念

竞争学习是自组织网络中最常用的一种学习策略，首先说明与之相关的几个基本概念。

1. 模式、分类、聚类与相似性

在神经网络应用中，输入样本、输入模式和输入模式样本这类术语经常混用。一般当神经网络涉及识别、分类问题时，常常用到输入模式的概念。模式是对某些感兴趣的客体的定量描述或结构描述，模式类是具有某些共同特征的模式集合。分类是在类别知识等导师信号的指导下，将待识别的输入模式分配到各自的模式类中去。无导师指导的分类称为聚类，聚类的目的是将相似的模式样本划归一类，而将不相似的分离开，其结果实现了模式样本的类内相似性和类间分离性。由于无导师学习的训练样本中不含有期望输出，因此对于某一输入模式样本应属于哪一类并没有任何先验知识。对于一组输入模式，只能根据它们之间的相似程度分为若干类，因此相似性是输入模式的聚类依据。

2. 相似性测量

神经网络的输入模式用向量表示，比较不同模式的相似性可转化为比较两个向量的距离，因而可用模式向量间的距离作为聚类判据。模式识别中常用到的两种聚类判据是欧式距离法和余弦法。

1) 欧式距离法

设 \mathbf{X}, \mathbf{X}_i 为两向量，其间的欧式距离：

$$d = \|\mathbf{X} - \mathbf{X}_i\| = \sqrt{(\mathbf{X} - \mathbf{X}_i)(\mathbf{X} - \mathbf{X}_i)^T} \quad (18-1)$$

d 越小， \mathbf{X} 与 \mathbf{X}_i 越接近，两者越相似，当 $d=0$ 时， $\mathbf{X} = \mathbf{X}_i$ ；以 $d=T$ （常数）为判据，可对输入向量模式进行聚类分析：

由于 d_{12}, d_{23}, d_{31} 均小于 T ， d_{45}, d_{56}, d_{46} 均小于 T ，而 $d_{1i} > T (i=4,5,6)$ ， $d_{2i} > T (i=4,5,6)$ ， $d_{3i} > T (i=4,5,6)$ ，于是将输入模式 $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \mathbf{X}_5, \mathbf{X}_6$ 分为类 1 和类 2 两大类，如图 18-2 所示。

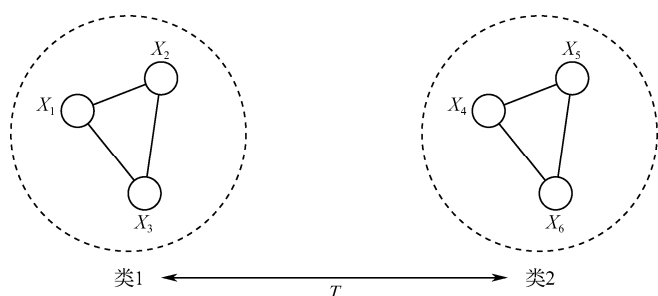


图 18-2 基于欧式距离法的模式分类

2) 余弦法

两个模式向量越接近，其夹角越小，余弦越大。当两个模式向量完全相同时，其余弦夹角为 1。如果对同一类内各个模式向量间的夹角作出规定，不允许超过某一最大夹角 φ ，则最大夹角就成为一种聚类判据。同类模式向量的夹角小于 φ ，两类模式向量的夹角大于 φ 。余弦法适合模式向量长度相同和模式特征只与向量方向相关的相似性测量，分类效果如图 18-3 所示。

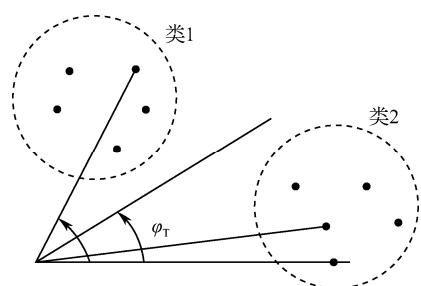


图 18-3 基于余弦的相似性测量

设 \mathbf{X}, \mathbf{X}_i 为两向量，其间的夹角余弦

$$\cos \varphi = \frac{\mathbf{X} \mathbf{X}_i^T}{\|\mathbf{X}\| \|\mathbf{X}_i\|} \quad (18-2)$$

φ 越小， \mathbf{X} 与 \mathbf{X}_i 越接近，两者越相似；当 $\varphi=0$ 时， $\cos \varphi=1$ ， $\mathbf{X} = \mathbf{X}_i$ ；同样以 $\varphi = \varphi_0$ 为判据可进行聚类分析。

18.2 竞争学习规则

实验表明，人眼的视网膜、脊髓和海马中存一种侧抑制现象，即当一个神经细胞兴奋后，会对其周围的神经细胞产生抑制作用。

这种侧抑制使神经细胞之间呈现出竞争，开始时可能多个细胞同时兴奋，但一个兴奋程度最强的神经细胞对周围神经细胞的抑制作用也最强，其结果使其周围神经细胞兴奋程度减弱，从而该神经细胞是这次竞争的“胜者”，其他神经细胞在竞争中失败。

图 18-1 的自组织网络（竞争型神经网络）构成的基本思想是网络的竞争层各神经元竞争



对输入模式响应的机会，最后仅有一个神经元成为竞争的“胜者”，这一获胜神经元则表示对输入模式的识别。体现了物生神经细胞的侧抑制竞争机制。

1) 向量归一化

对自组织网络中的当前输入模式向量 \mathbf{X} 、竞争层中各神经元对应的内星权向量 \mathbf{W}_j ($j=1,2,\dots,m$)，全部进行归一化处理，如图 18-4 示，得到 $\hat{\mathbf{X}}$ 和 $\hat{\mathbf{W}}_j$ ：

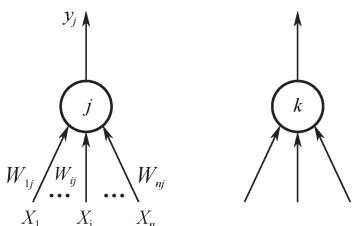


图 18-4 向量归一化

$$\hat{\mathbf{X}} = \frac{\mathbf{X}}{\|\mathbf{X}\|}, \quad \hat{\mathbf{W}}_j = \frac{\mathbf{W}_j}{\|\mathbf{W}_j\|} \quad (18-3)$$

2) 寻找获胜神经元

将 $\hat{\mathbf{X}}$ 与竞争层所有神经元对应的内星权向量 $\hat{\mathbf{W}}_j$ ($j=1,2,\dots,m$) 进行相似性比较。最相似的神经元获胜，权向量为 $\hat{\mathbf{W}}_{j^*}$ ：

$$\begin{aligned} \|\hat{\mathbf{X}} - \hat{\mathbf{W}}_{j^*}\| &= \min_{j \in \{1,2,\dots,n\}} \{\|\hat{\mathbf{X}} - \hat{\mathbf{W}}_j\|\} \\ \Rightarrow \|\hat{\mathbf{X}} - \hat{\mathbf{W}}_{j^*}\| &= \sqrt{(\hat{\mathbf{X}} - \hat{\mathbf{W}}_{j^*})(\hat{\mathbf{X}} - \hat{\mathbf{W}}_{j^*})^T} = \sqrt{\hat{\mathbf{X}}\hat{\mathbf{X}}^T - 2\hat{\mathbf{W}}_{j^*}\hat{\mathbf{X}}^T + \hat{\mathbf{W}}_{j^*}\hat{\mathbf{W}}_{j^*}^T} \\ &= \sqrt{2(1 - \hat{\mathbf{W}}_{j^*}\hat{\mathbf{X}}^T)} \Rightarrow \hat{\mathbf{W}}_{j^*}\hat{\mathbf{X}}^T = \max_j (\hat{\mathbf{W}}_j\hat{\mathbf{X}}^T) \end{aligned} \quad (18-4)$$

3) 网络输出与权调整

按 WTA 学习法则，获胜神经元输出为“1”，其余为 0，即

$$y_j(t+1) = \begin{cases} 1 & j = j^* \\ 0 & j \neq j^* \end{cases} \quad (18-5)$$

只有获胜神经元才有权调整其权向量 \mathbf{W}_{j^*} ，其权向量学习调整如下：

$$\begin{cases} \mathbf{W}_{j^*}(t+1) = \hat{\mathbf{W}}_{j^*}(t) + \Delta \mathbf{W}_{j^*} = \hat{\mathbf{W}}_{j^*}(t) + \alpha(\hat{\mathbf{X}} - \hat{\mathbf{W}}_{j^*}) \\ \mathbf{W}_j(t+1) = \hat{\mathbf{W}}_j(t) & j \neq j^* \end{cases} \quad (18-6)$$

$0 < \alpha \leq 1$ 为学习率， α 一般随着学习的进展而减小，即调整的程度越来越小，趋于聚类中心。

4) 重新归一化处理

归一化后的权向量经过调整后，得到的新向量不再是单位向量，因此要对学习调整后的向量重新进行归一化，循环运算，直到学习率 α 衰减到 0。

18.3 竞争学习原理

设输入模式为二维向量，归一化后其矢端可以看成分布在图 18-5 所示的单位圆上的点，



用“O”表示。设竞争层有 4 个神经元，对应的 4 个内星权向量归一化后也标在同一单位圆上，用“*”表示。从输入模式点的分布可看出，它们大体上聚集为 4 个簇，因而可以分为 4 类。然而自组织网络的训练样本中只提供了输入模式而没有提供关于分类的指导信息，网络是如何通过竞争机制自动发现样本空间的类别划分呢？

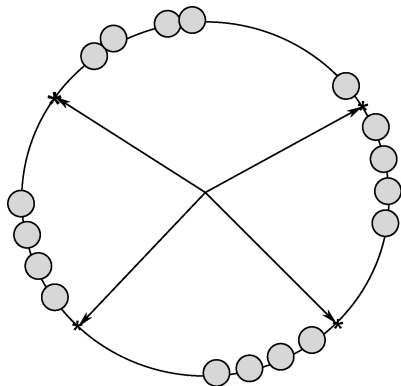


图 18-5 竞争学习的几何意义

自组织网络在开始训练前先对竞争层的权向量进行随机初始化。因此在初始状态时，单位圆上的“*”是随机分布的。两个等长向量的点积越大，两者越近似，因此以点积最大获胜的神经元对应权向量应最接近当前输入模式。从图 18-5 可看出，如果当前输入模式用空心圆“O”表示，单位圆上各“*”点代表的权向量依次同“O”点代表的输入向量比较距离，结果是离得最近的那个“*”点获胜。从获胜神经元的权值调整可看出，调整的结果是使 W_j 进一步接近当前输入 X_o 。这一点从图 18-6 的向量合成图上可看出。调整后，获胜“*”点的位置进一步移向“O”点及其所在的簇。显然，当下次出现与“O”相像的同簇内的输入模式时，上次获胜的“*”点更容易获胜。依此方式经过充分训练后，单位圆上的 4 个“*”点会逐渐移入各输入模式的簇中心，从而使竞争层每个神经元的权向量成为一类输入模式的聚类中心。当向网络输入一个模式时，竞争层中哪个神经元获胜使输出为 1，当前输入模式就归为哪类。

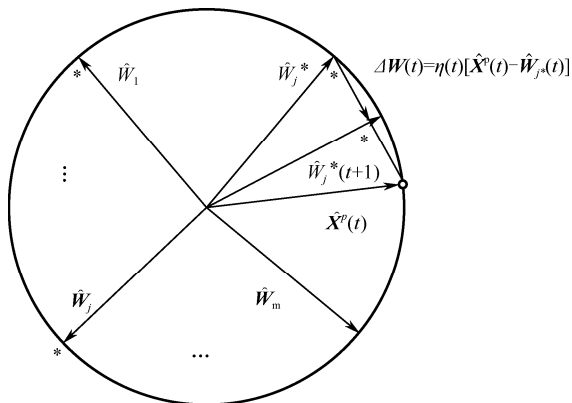


图 18-6 自组织权向量调整图

【例 18-1】 用竞争学习算法将下列各模式分为两类：



$$\mathbf{X}_1 = \begin{pmatrix} 0.8 \\ 0.6 \end{pmatrix}, \quad \mathbf{X}_2 = \begin{pmatrix} 0.1736 \\ -0.9848 \end{pmatrix}, \quad \mathbf{X}_3 = \begin{pmatrix} 0.707 \\ 0.707 \end{pmatrix}, \quad \mathbf{X}_4 = \begin{pmatrix} 0.342 \\ -0.9397 \end{pmatrix}, \quad \mathbf{X}_5 = \begin{pmatrix} 0.6 \\ 0.8 \end{pmatrix}$$

解：将上述输入模式转换为极坐标形式 $\mathbf{X}_1 = 1 \angle 36.89^\circ$, $\mathbf{X}_2 = 1 \angle -80^\circ$, $\mathbf{X}_3 = 1 \angle 45^\circ$, $\mathbf{X}_4 = 1 \angle -70^\circ$, $\mathbf{X}_5 = 1 \angle 53.13^\circ$, 如图 18-7 所示。

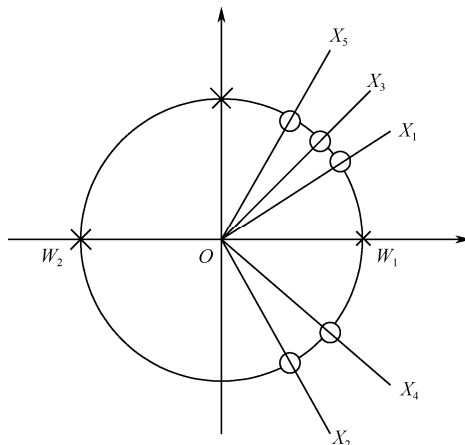


图 18-7 模式向量图

竞争层设两个权向量，随机初始化为单位向量：

$$\mathbf{W}_1(0) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1 \angle 0^\circ, \quad \mathbf{W}_2(0) \begin{pmatrix} -1 \\ 0 \end{pmatrix} = 1 \angle 180^\circ$$

取学习率 $\eta = 0.5$ ，按 1~5 的顺序依次输入模式向量，用式 (18-6) 给出的算法调整权值，每次修改后重新进行归一化。前 20 次训练中两个权向量的变化情况如表 18-1 所示。

表 18-1 权向量调整过程

训练次数	\mathbf{W}_1	\mathbf{W}_2	训练次数	\mathbf{W}_1	\mathbf{W}_2
1	18.43°	-180°	11	40.5°	-100°
2	-30.8°	-180°	12	40.5°	-90°
3	7°	-180°	13	43°	-90°
4	-32°	-180°	14	43°	-81°
5	11°	-180°	15	47.5°	-81°
6	24°	-180°	16	42°	-81°
7	24°	-130°	17	42°	-80.5°
8	34°	-130°	18	43.5°	-80.5°
9	34°	-100°	19	43.5°	-75°
10	44°	-100°	20	48.5°	-75°

如将 5 个输入模式和 2 个权向量标在单位圆中，可以明显看出， \mathbf{X}_1 、 \mathbf{X}_2 、 \mathbf{X}_5 属于同一



模式类，其中心向量应为 $\frac{1}{3}(\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_5) = 1\angle 45^\circ$ ； \mathbf{X}_2 、 \mathbf{X}_4 属于同一模式类，其中心向量

为 $\frac{1}{2}(\mathbf{X}_2 + \mathbf{X}_4) = 1\angle -75^\circ$ 。经过 20 次训练， \mathbf{W}_1 和 \mathbf{W}_2 就已经非常接近 $1\angle 45^\circ$ 和 $1\angle -75^\circ$ 了。

如果训练一直继续，两个权向量是否会最终收敛于两个模式类中心？事实上，如果训练中学习率保持为常数， \mathbf{W}_1 和 \mathbf{W}_2 将在 $1\angle 45^\circ$ 和 $1\angle -75^\circ$ 附近摆动，永远也不可能收敛。只有当学习率随训练时间不断下降，才有可能使摆动减弱至终止。

18.4 竞争神经网络 MATLAB 实现

在 MATLAB 神经网络工具箱中提供了相关函数实现竞争神经网络的创建、学习等。

1. 创建函数 newc

该函数用于建立一个竞争层神经网络，其调用格式如下：

```
net = newc(PR,S,KLR,CLR)
```

其中，

PR 为一个 $R \times 2$ 维的输入矩阵，它决定了输入向量的最小值和最大值的取值范围；

R 为输入向量的个数；

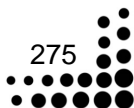
S 为神经元的个数；KLR 为 Kohonen 学习速率，默认值为 0.01；

CLR 为 Conscience 学习速率，默认值为 0.001；

net 为生成的自组织竞争网络。

【例 18-2】 建立一个输入向量分布在一个二维空间，其变化范围分别为[0 1]和[0 1]，用来区分 5 种模式的自组织竞争网络，并对其进行训练和仿真。

```
>> clear all;
C=[0 1;0 1];
clusters=5; %产生具有 5 类样本类别的样本点,并在图中绘制出来
points=10; std_dev=0.05;
X=nngenc(C,clusters,points,std_dev);
plot(X(1,:),X(2,:),'rp');
hold on;
xlabel('X(1)'); ylabel('X(2)');
% 建立神经元为 5 的一个自组织竞争神经网络,并求出初始权值
net=newc([0 1;0 1],5,0.1);
w=net.iw{1}; %初始权值
% 训练神经网络,并设置最大训练步数为 7
net.trainParam.epochs=7; %最大训练步数
net=init(net);
net=train(net,X);
w1=net.iw{1}; %训练后的权值
```



```
plot(X(1,:),X(2,:),'bo');
xlabel('X(1)'); ylabel('X(2)');
hold on;
plot(w1(:,1),w1(:,2),'g+');
hold off;
% 对于训练好的网络进行测试与使用
x1=[0.6;0.8];
y=sim(net,x1)
```

运行程序，其输出结果如下，训练记录过程如图 18-8 所示，训练后网络权值的分布如图 18-9 所示。

```
y =
(5,1)      1
```

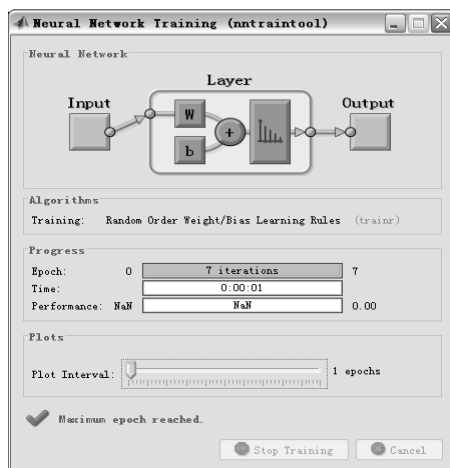


图 18-8 训练过程记录图

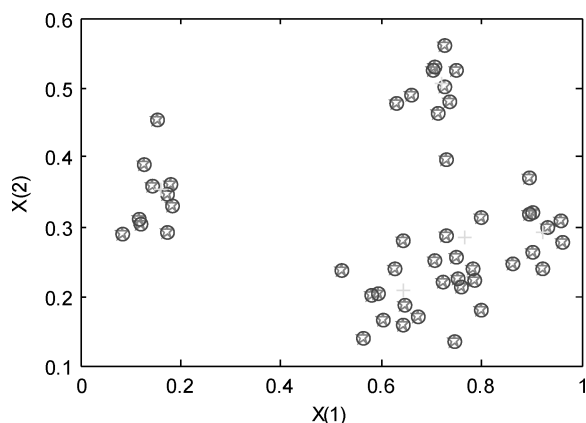


图 18-9 训练后网络权值的分布

从图 18-9 中可以看出，网络经过训练后，权值得到了调整，调整后的权值分布在各类的中心位置上。对于需要分类的模式向量[0.6; 0.8]，将其输入到训练好的网络中，网络就可以对



其分类。分类结果指出了第 (5,1) 个神经元发生了响应,这反映了这个输入所属的类别。

【例 18-3】 以竞争型神经网络完成如图 18-10 所示的三类模式的分类。

$$p_1 = \begin{bmatrix} -0.1961 \\ 0.9806 \end{bmatrix}, p_2 = \begin{bmatrix} 0.1961 \\ 0.9806 \end{bmatrix}, p_3 = \begin{bmatrix} 0.9806 \\ 0.1961 \end{bmatrix}$$

$$p_4 = \begin{bmatrix} 0.9806 \\ -0.1961 \end{bmatrix}, p_5 = \begin{bmatrix} -0.5812 \\ -0.8137 \end{bmatrix}, p_6 = \begin{bmatrix} -0.8137 \\ -0.5812 \end{bmatrix}$$

解: 图中的三类模式从它们在二维平面上的位置特征看具有明显的分类特征, 可以以竞争型神经网络完成其分类。

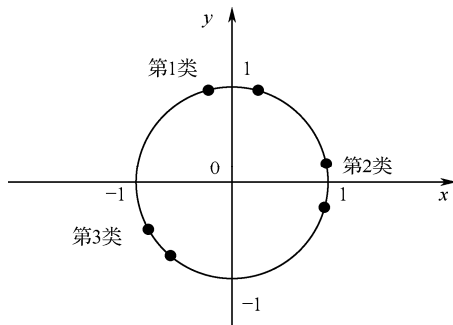


图 18-10 待分类模式

本例实现的 MATLAB 程序为:

```
>>clear all
%定义输入向量
P=[-0.1961 0.1961 0.9806 0.9806 -0.5812 -0.8137;
    0.9806 0.9806 0.1961 -0.1961 -0.8137 -0.5812];
%创建竞争型网络
net=newc([-1 1;-1 1],3);
%训练神经网络
net=train(net,P);
%定义待测试样本输入向量
p=[-0.1961 0.1961 0.9806 0.9806 -0.5812 -0.8137;
    0.9806 0.9806 0.1961 -0.1961 -0.8137 -0.5812];
%网络仿真
y=sim(net,p);
%输出仿真结果
yc=vec2ind(y)
```

仿真结果为:

```
yc =
     2     2     1     1     3     3
```

结果很好地完成了分类。



2. 学习函数 learnk

learnk 函数根据 Konohen 相关准则计算网络层的权值变化矩阵，其学习通过调整神经元的权值等于当前输入，使神经元存储输入，用于以后的识别， $\Delta w(i, j) = \eta(x(j) - w(i, j))$ 。该函数的调用格式如下：

```
[dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnk(code)
```

式中，

W 为 $S \times R$ 维的权值矩阵（或 $S \times 1$ 维的阈值向量）；

P 为 Q 组 R 维的输入向量矩阵（或 Q 组单个输入）；

Z 为 Q 组 S 维的权值输入向量；

N 为 Q 组 S 维的网络输入向量；

A 为 Q 组 S 维的输出向量；

T 为 Q 组 S 维的目标向量；

E 为 Q 组 S 维的网络误差向量（ $E=T-Y$ ，T 为网络的目标向量，Y 为网络的输出向量）；

gW 为 $S \times R$ 维的性能参数的梯度；

gA 为 Q 组 S 维的性能参数的输出梯度；

LP 为学习参数，如果没有则为空；

D 为 $S \times R$ 维的神经元间距离；

LS 为学习状态，初始值为空；

在函数返回中，dW 为 $S \times R$ 维权值变化矩阵，LS 为新的学习状态。

Info = learnk(code)：针对不同的 code 返回相应的有用信息，包括：

- pnames——返回学习参数的名称；
- pdefaults——返回默认的学习参数；
- needg——如果函数使用了 gW 或 gA，则返回 1。

【例 18-4】 learnis 函数应用示例。

```
>> clear all;
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.lr = 0.5;
dW = learnis(w,p,[],[],a,[],[],[],[],lp,[])
```

运行程序，输出如下：

```
dW =
    0.2045   -0.1325
    0.1655    0.0447
   -0.0766    0.1177
```



3. 传递 compet 函数

该函数为竞争性传递函数，用于神经元的网络输入转换，其调用格式为如下所述。

$A = \text{compet}(N, FP)$: N 为 $S \times Q$ 维的网络输入（列）向量矩阵， FP 为性能参数（可忽略）返回网络输入向量 N 的输出矩阵 A 。

$dA_dN = \text{compet}('dn', N, A, FP)$: 返回 A 关于 N 的导数 dA_dN ，如果 A 或 FP 没有给出或为空矩阵，则 FP 返回默认参数。

$\text{info} = \text{compet}(\text{code})$: 依据 code 值的不同，返回不同的信息。

- $\text{compet}('name')$ ——返回传输函数最小、最大值的有效输入范围；
- $\text{compet}('output', FP)$ ——返回传输函数的全称；
- $\text{compet}('active', FP)$ ——返回传输函数最小、最大值的二元向量；
- $\text{compet}('fulldderiv')$ ——根据 dA_dN 是 $S \times S \times Q$ 还是 $S \times Q$ 来确定返回 1 还是 0；
- $\text{compet}('fpnames')$ ——返回函数参数的名称；
- $\text{compet}('fpdefaults')$ ——返回默认的函数参数。

18.5 竞争型神经网络存在的问题

对于模式样本本身具有较明显的分类特征，竞争型神经网络可以对其进行正确的分类，网络对同一类或相似的输入模式具有较稳定的输出响应，但也存在一些问题：

（1）当学习模式样本本身杂乱无章，没有明显的分类特征时，网络对输入模式的响应呈现振荡的现象，即对同一类输入模式的响应可能激活不同的输出神经元，从而不能实现正确的分类。当各类模式的特征相近时，也会出现同样的情况。

（2）在权值和阈值的调整过程中，学习率的选择在收敛速度和稳定性之间存在矛盾，而不像前面介绍的其他学习算法，可以在刚开始时采用较大的学习率，而在权值和阈值趋于稳定时，采用较小的学习率。竞争型神经网络在增加新的学习样本时，对权值和阈值可能需要做比前一次更大的调整。

（3）网络的分类性能与权值和阈值的初始值、学习率、训练样本的顺序、训练时间的长短（训练次数）等都有关系，而目前还没有有效的方法对各种因素的影响进行评判。

（4）在 MATLAB 神经网络工具箱中，以函数 trainr 进行竞争型神经网络的训练，用户只能限定训练的最长时间或训练的最大次数，以此终止训练，但终止训练时网络的分类性能究竟如何，没有明确的评判指标。



第 19 章 Elman 网络算法分析与应用

Elman 网络是 J. L. Elman 于 1990 年首先针对语音处理问题提出来的,它是一种典型的局部回归网络。Elman 网络可以看作一个具有局部记忆单元和局部反馈连接的前向神经网络。Elman 网络具有与多层前向网络相似的多层结构。它的主要结构是前馈连接,包括输入层、隐含层、输出层,其连接权可以进行学习修正;反馈连接由一组“结构”单元构成,用来记忆前一时刻的输出值,其连接权值是固定的。在这种网络中,除了普通的隐含层外,还有一个特别的隐含层,称为关联层(或联系单元层);该层从隐含层接收反馈信号,每一个隐含层节点都有一个与之对应的关联层节点连接。关联层的作用是通过连接记忆将上一个时刻的隐含层状态连同当前时刻的网络输入一起作为隐含层的输入,相当于状态反馈。隐含层的传递函数仍为某种非线性函数,一般为 Sigmoid 函数,输出层为线性函数,关联层也为线性函数。

19.1 Elman 神经网络结构

Elman 型回归神经网络一般分为 4 层:输入层、中间层(隐含层)、承接层和输出层,如图 19-1 所示。其输入层、隐含层和输出层的连接类似于前馈网络,输入层的单元仅起信号传输作用,输出层单元起线性加权作用。隐含层单元的传递函数可采用线性或非线性函数,承接层又称为上下文层或状态层,它用来记忆隐含层单元前一时刻的输出值并返回给输入,可以认为是一个一步延时算子。

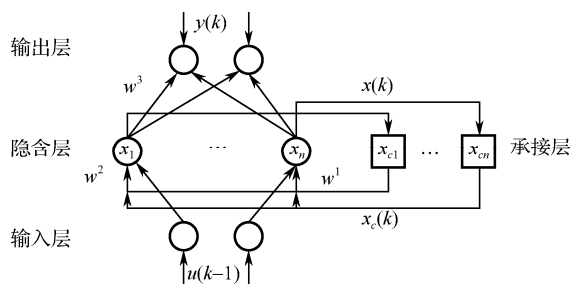


图 19-1 Elman 神经网络的模型

Elman 型回归神经网络的特点是隐含层的输出通过承接层的延迟与存储,自联到隐含层的输入。这种自联方式使其对历史状态的数据具有敏感性,内部反馈网络的加入增加了网络本身处理动态信息的能力,从而达到了动态建模的目的。



19.2 Elman 神经网络权值修正的学习算法

对于递归网络,有时为了计算的简便,在修正权值时可以采用静态的 BP 算法。但由于网络的输出不仅与 k 时刻的输入有关,而且还与 k 以前时刻的输入信号有关,所以涉及精确计算时,必须采用动态的学习规则。

在前身 BP 网络中,对学习算法推导时,采用的是链式法则的算法。而在对递归网络算法的研究中则有所不同,一般采用有序链式法则的算法。此外,对递归网络进行训练时,一共有两种方法:一种是批处理模式,另一种是在线模式,本节采用后者,即在线训练规则。

定义 k 时刻网络权值调整的误差函数 E :

$$E(k) = \frac{1}{2} \sum_{i=1}^r (d_i(k) - y_i(k))^2 \quad (19-1)$$

其中, $d_i(k)$ 为 k 时刻第 i 个输出节点的期望输出。

令 $e_i(k)$ 为 k 时刻第 i 个输出节点期望输出与实际输出的误差,即

$$e_i(k) = d_i(k) - y_i(k) \quad (19-2)$$

网络的权值变化为:

$$w(k+1) = w(k) + \eta \left(-\frac{\partial E(k)}{\partial w} \right) + \alpha \Delta w(k) \quad (19-3)$$

其中 w 可代表输入层、隐含层或输出层的权值。

对于输出层的权值,采用有序链式法则,有:

$$\begin{aligned} -\frac{\partial E(k)}{\partial w_{ij}^1} &= -\frac{\partial E(k)}{\partial y_i(k)} \cdot \frac{\partial y_i(k)}{\partial w_{ij}^1} \\ &= -\frac{\partial E(k)}{\partial y_i(k)} \cdot \frac{\partial y_i(k)}{\partial s_i^3(k)} \cdot \frac{\partial s_i^3(k)}{\partial w_{ij}^1} = e_i(k) \cdot f'2'(s_i^3(k)) \cdot x_j^1(k) \end{aligned} \quad (19-4)$$

对于隐含层的权值,同理有:

$$\begin{aligned} -\frac{\partial E(k)}{\partial w_{ij}^0} &= -\sum_{l=1}^r \frac{\partial E(k)}{\partial y_l(k)} \cdot \frac{\partial y_l(k)}{\partial w_{ij}^0} \\ &= -\sum_{l=1}^r \frac{\partial E(k)}{\partial y_l(k)} \cdot \frac{\partial y_l(k)}{\partial s_l^3(k)} \cdot \frac{\partial s_l^3(k)}{\partial x_i^1(k)} \cdot \frac{\partial x_i^1(k)}{\partial w_{ij}^0} \\ &= \sum_{l=1}^r e_l(k) \cdot f'2'(s_l^3(k)) \cdot w_{li}^1(k) \cdot \frac{\partial x_i^1(k)}{\partial w_{ij}^0} \end{aligned} \quad (19-5)$$

若令 $\beta_{ij}^i(k) = \frac{\partial x_i^1(k)}{\partial w_{ij}^0}$, 则有:



$$\begin{aligned}
\beta_{ij}^i(k) &= \frac{\partial x_i^1(k)}{\partial w_{ij}^0} = \frac{\partial x_i^1(k)}{\partial s_i^1(k)} \cdot \frac{\partial s_i^1(k)}{\partial w_{ij}^0} \\
&= f1'(s_i^1(k)) \cdot \left(x_j^0(k) + \sum_{m=1}^{n^1} w_{im}^2 \cdot \frac{\partial c_m(k)}{\partial w_{ij}^0} \right) \\
&= f1'(s_i^1(k)) \cdot \left(x_j^0(k) + \sum_{m=1}^{n^1} w_{im}^2 \cdot \frac{\partial x_m^1(k-1)}{\partial w_{ij}^0} \right) \\
&= f1'(s_i^1(k)) \cdot \left(x_j^0(k) + \sum_{m=1}^{n^1} w_{im}^2 \cdot \beta_{ij}^m(k-1) \right)
\end{aligned} \tag{19-6}$$

那么可以得到:

$$\begin{cases} -\frac{\partial E(k)}{\partial w_{ij}^0} = \sum_{l=1}^r e_l(k) \cdot f2'(s_l^3(k)) \cdot w_{li}^1(k) \cdot \beta_{ij}^i(k) \\ \beta_{ij}^i(k) = f1'(s_i^1(k)) \cdot \left(x_j^0(k) + \sum_{m=1}^{n^1} w_{im}^2 \cdot \beta_{ij}^m(k-1) \right) \end{cases} \tag{19-7}$$

其中, $\beta_{ij}^m(0) = 0$; $m, i, j = 1, 2, \dots, n$ 。

同理, 对关联层的权值, 我们有:

$$\begin{cases} -\frac{\partial E(k)}{\partial w_{ij}^2} = \sum_{l=1}^r e_l(k) \cdot f2'(s_l^3(k)) \cdot w_{li}^1(k) \cdot \delta_{ij}^i(k) \\ \delta_{ij}^i(k) = f1'(s_i^1(k)) \cdot \left(x_j^1(k-1) + \sum_{m=1}^{n^1} w_{im}^2 \cdot \delta_{ij}^m(k-1) \right) \end{cases} \tag{19-8}$$

其中, $\delta_{ij}^m(0) = 0$; $m, i, j = 1, 2, \dots, n$ 。

19.3 Elman 网络稳定性推导

在此将采用 Lyapunov 稳定性理论来分析所研究的 Elman 网络的稳定性。采用全局误差函数的定义式 (19-2) 作为李氏函数, 将式 (19-2) 代入式 (19-1), 有:

$$E(k) = \frac{1}{2} \sum_{i=1}^r e_i^2(k) \tag{19-9}$$

要保证系统的稳定性, 必须要有:

$$\Delta E(k+1) = E(k+1) - E(k) < 0 \tag{19-10}$$

即:

$$\frac{1}{2} \sum_{i=1}^r (e_i^2(k+1) - e_i^2(k)) < 0 \tag{19-11}$$

当网络权值变化较小时, 对 $e_i(k+1)$ 进行 Taylor 展开, 有:

$$e_i(k+1) = e_i(k) + \frac{\partial e_i(k)}{\partial w} \cdot \Delta w + \dots \approx e_i(k) + \frac{\partial e_i(k)}{\partial w} \cdot \Delta w \tag{19-12}$$



将式 (19-12) 代入式 (19-11), 则有:

$$\begin{aligned}\Delta E(k+1) &= \frac{1}{2} \sum_{i=1}^r \left(e_i^2(k) + 2 \frac{\partial e_i(k)}{\partial w} \cdot \Delta w \cdot e_i(k) + \left(\frac{\partial e_i(k)}{\partial w} \cdot \Delta w \right)^2 - e_i^2(k) \right) \\ &= \frac{1}{2} \sum_{i=1}^r \frac{\partial e_i(k)}{\partial w} \cdot \Delta w \cdot \left(2e_i(k) + \frac{\partial e_i(k)}{\partial w} \cdot \Delta w \right)\end{aligned}\quad (19-13)$$

在权值变化较小的情况下, 我们可以对式 (19-13) 进行近似:

$$\Delta w \approx \eta \left(-\frac{\partial E(k)}{\partial w} \right) = -\eta \sum_{i=1}^r \frac{\partial E(k)}{\partial e_j(k)} \cdot \frac{\partial e_j(k)}{\partial w} = -\eta \sum_{j=1}^r e_j(k) \cdot \frac{\partial e_j(k)}{\partial w} \quad (19-14)$$

把式 (19-14) 代入式 (19-13), 可得:

$$\begin{aligned}\Delta E(k+1) &= -\frac{1}{2} \sum_{i=1}^r \frac{\partial e_i(k)}{\partial w} \cdot \Delta w \cdot \left(2e_i(k) + \frac{\partial e_i(k)}{\partial w} \cdot \left(-\eta \sum_{j=1}^r e_j(k) \cdot \frac{\partial e_j(k)}{\partial w} \right) \right) \\ &= -\frac{1}{2} \sum_{i=1}^r \frac{\partial e_i(k)}{\partial w} \cdot \left(-\eta \sum_{j=1}^r e_j(k) \cdot \frac{\partial e_j(k)}{\partial w} \right) \cdot \left(2e_i(k) + \frac{\partial e_i(k)}{\partial w} \cdot \left(-\eta \sum_{j=1}^r e_j(k) \cdot \frac{\partial e_j(k)}{\partial w} \right) \right) \\ &= \frac{1}{2} \sum_{i=1}^r \sum_{j=1}^r e_j(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w} \cdot \left(2e_i(k) - \eta \cdot \left(\sum_{j=1}^r e_j(k) \cdot \frac{\partial e_j(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w} \right) \right) \\ &= -\frac{1}{2} \eta \left(\eta \cdot \sum_{i=1}^r \sum_{j=1}^r e_j(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w} \right)^2 - 2 \sum_{i=1}^r \sum_{j=1}^r e_i(k) \cdot e_j(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w}\end{aligned}\quad (19-15)$$

根据式 (19-11), 得:

$$0 < \eta < \frac{2 \sum_{i=1}^r \sum_{j=1}^r e_i(k) \cdot e_j(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w}}{\sum_{i=1}^r \sum_{j=1}^r \left(e_i(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w} \right)^2} \quad (19-16)$$

令:

$$g_{\max} = \max \left\| \frac{\sum_{i=1}^r \sum_{j=1}^r \left(e_j(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w} \right)^2}{\sum_{i=1}^r \sum_{j=1}^r e_i(k) \cdot e_j(k) \cdot \frac{\partial e_i(k)}{\partial w} \cdot \frac{\partial e_j(k)}{\partial w}} \right\| \quad (19-17)$$

于是有:

$$0 < \eta < \frac{2}{g_{\max}} \quad (19-18)$$

式 (19-17) 或式 (19-18) 就是保证 Elman 网络稳定学习速率的取值范围。

19.4 对角递归网络稳定时学习速率的确定

根据式 (19-16), 对角网络是取 $i=1, j=1$ 的情况, 此时可以得到网络稳定的学习速率的



取值范围。

$$0 < \eta < \frac{2}{\left(\frac{\partial e(k)}{\partial w}\right)^2} \quad (19-19)$$

(1) 如果令 $g_{\max}^D = \max_k \left\| \frac{\partial e(k)}{\partial w} \right\|$ ，则有：

$$0 < \eta < \frac{2}{(g_{\max}^D)^2} \quad (19-20)$$

(2) 若令 $f_1(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$ ， $f_2(x) = x$ ，可以求得输出层、关联层及隐含层的学习速率取值范围分别为：

$$\begin{aligned} 0 < \eta^O &< \frac{2}{n^1} \\ 0 < \eta^D &< \frac{2}{n^1 (w_{\max}^1)^2} \\ 0 < \eta^I &< \frac{2}{n^1 (w_{\max}^1 x_{\max}^0)^2} \end{aligned} \quad (19-21)$$

之所以得到这样的结果，与 $\frac{\partial e(k)}{\partial w}$ 的取值有关。限于篇幅，详细的推导请查阅相关的资料。

在实际应用中，各层的学习速率分别取最佳值为：

$$\begin{aligned} \eta_{opt}^O &= \frac{1}{n^1} \\ \eta_{opt}^D &= \frac{2}{n^1 (w_{\max}^1)^2} \\ \eta_{opt}^I &= \frac{2}{n^1 (w_{\max}^1 x_{\max}^0)^2} \end{aligned}$$

19.5 Elman 神经网络在数据预测中的应用

电力系统由电力网、电力用户共同组成，其任务是给广大用户不间断地提供经济、可靠、符合质量标准的电能，满足各类负荷的需求，为社会发展提供动力。由于电力的生产与使用具有特殊性，即电能难以大量储存，而且各类用户对电力的需求是时刻变化的，这就要求系统发电出力应随时紧跟系统负荷的变化动态平衡，即系统要最大限度地发挥设备能力，使整个系统保持稳定且高效地运行，以满足用户的需求。否则，就会影响供电、用电的质量，甚至危及系统的安全与稳定。因此，电力系统负荷预测技术发展起来，并且是这一切得以顺利进行的前提和基础。

负荷预测的核心问题是预测的技术问题，或者说是预测的数学模型。传统的数学模型是用现成的数学表达式加以描述，具有计算量小、速度快的优点，但同时也存在很多的缺陷和



局限性,比如不具备自学习、自适应能力、预测系统的鲁棒性没有保障等。特别是随着我国经济的发展,电力系统的结构日趋复杂,电力负荷变化的非线性、时变性和不确定性的特点更加明显,很难建立一个合适的数学模型来清晰地表达负荷和影响负荷的变量之间的关系,而基于神经网络的非数学模型预测法为解决数学模型法的不足提供了新的思路。

利用人工神经网络对电力系统负荷进行预测,实际上是利用人工神经网络可以任意逼近任一非线性函数的特性及通过学习历史数据建模的优点。而在各种人工神经网络中,反馈式神经网络又因为其具有输入延迟,进而适合应用于电力系统负荷预测。根据负荷的数据,选定反馈神经网络的输入、输出节点,来反映电力系统负荷运行的内在规律,从而达到预测未来时段负荷的目的。因此,用人工神经网络对电力系统负荷进行预测,首要的问题是确定神经网络的输入、输出节点,能使其反映电力负荷的运行规律。

【例 19-1】一般来说,电力系统的高峰通常出现在每天的 9~19 时之间,本例只对每天上午的逐时负荷进行预测,即预测每天 9~11 时共 3h 负荷数据。电力系统负荷数据如表 19-1 所列。以下数据为真实数据,已经过归一化。

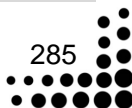
表 19-1 电力系统负载数据

时 间	负 荷 数 据			时 间	负 荷 数 据		
2008.10.10	0.1291	0.4842	0.7976	2008.10.15	0.1719	0.6011	0.754
2008.10.11	0.1084	0.4579	0.8187	2008.10.16	0.1237	0.4425	0.8031
2008.10.12	0.1828	0.7977	0.743	2008.10.17	0.1721	0.6252	0.7626
2008.10.13	0.122	0.5468	0.8084	2008.10.18	0.1432	0.5845	0.7942
2008.10.14	0.113	0.3636	0.814				

利用前 8 天的数据作为网络的训练样本,每 3 天的负荷作为输入向量,第 4 天的负荷作为目标向量。这样可以得到 5 组训练样本。第 9 天的数据作为网络的测试样本,验证网络能否合理预测当前的负荷数据。

其实现的 MATLAB 代码为:

```
>> clear all;
a=[0.1291 0.4842 0.7976;0.1084 0.4579 0.8187;0.1828 0.7977 0.743;...
    0.122 0.5468 0.8048;0.113 0.3636 0.814;0.1719 0.6011 0.754;...
    0.1237 0.4425 0.8031;0.1721 0.6152 0.7626;0.1432 0.5845 0.7942];
for i=1:6
    p(i,:)=a(i,:),a(i+1,:),a(i+2,:);
end
% 训练数据输入
p_train=p(1:5,:);
% 训练数据输出
t_train=a(4:8,:);
% 测试数据输入
p_test=p(6,:);
% 测试数据输出
```





```
t_test=a(9,:);
% 为适应网络结构做转置
p_train=p_train';
t_train=t_train';
p_test=p_test';
% 网络的建立和训练
% 利用循环，设置不同的隐藏层神经元个数
nn=[7 11 14 18];
for i=1:4
    threshold=[0 1;0 1;0 1;0 1;0 1;0 1;0 1;0 1];
    % 建立 Elman 神经网络隐藏层为 nn(i)个神经元
    net=newelm(threshold,[nn(i),3],{'tansig','purelin'});
    % 设置网络训练参数
    net.trainparam.epochs=1000;
    net.trainparam.show=20;
    % 初始化网络
    net=init(net);
    % Elman 网络训练
    net=train(net,p_train,t_train);
    % 预测数据
    y=sim(net,p_test);
    % 计算误差
    error(i,:)=y'-t_test;
end

% 通过作图，观察不同隐含层神经元个数时，网络的预测效果
plot(1:1:3,error(1,:),'-ro');
hold on;
plot(1:1:3,error(2,:),'b:x');
hold on;
plot(1:1:3,error(3,:),'k-s');
hold on;
plot(1:1:3,error(4,:),'c--d');
hold on;
title('Elman 预测误差图');
set(gca,'Xtick',[1:3]);
legend('7','11','14','18');
xlabel('时间点');ylabel('误差');
hold off;
```

运行程序，效果如图 19-2 所示。

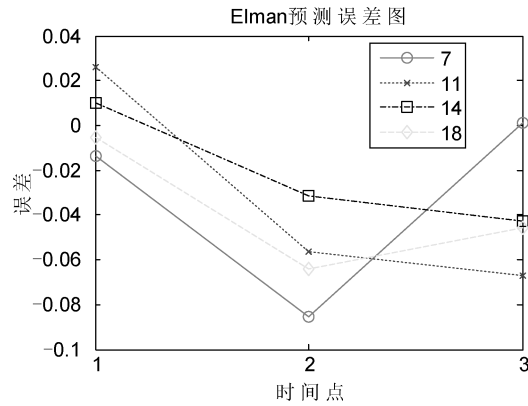
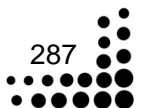


图 19-2 Elman 神经网络预测误差图

由图 19-2 可看出，网络预测误差还是比较小的，但是，中间神经元为 14 时出现了较大的误差。这可能是训练样本太小导致的。当中间神经元为 14 时，网络的预测误差最小，也就是预测性能最好。因此，于本例中中间层神经元的最佳数目应该为 14 个。



第 20 章 BP 网络工具箱函数及其应用

BP 神经网络是 1986 年由 Rumelhart 和 McClland 为首的科学家小组提出的,是一种按误差逆传播算法训练的多层前馈网络,是目前应用最广泛的神经网络模型之一。BP 神经网络由信息的正向传播和误差的反向传播两个过程组成。输入层各神经元负责接收来自外界的输入信息,并传递给中间层各神经元;中间层是内部信息处理层,负责信息变换,根据信息变换能力的需求,中间层可以设计为单隐层或者多隐层结构;最后一个隐层传递到输出层各神经元的信息经进一步处理后,完成一次学习的正向传播处理过程,由输出层向外界输出信息处理结果。当实际输出与期望输出不符时,进入误差的反向传播阶段。误差通过输出层,按误差梯度下降的方式修正各层权值,向隐含层、输入层逐层反传。周而复始的信息正向传播和误差反向传播过程是各层权值不断调整的过程,也是神经网络学习训练的过程,此过程一直进行到网络输出的误差减少到可以接受的程度,或者达到预先设定的学习次数为止。

BP 网络主要用于以下功能。

- 函数逼近:用输入向量和相应的输出向量训练一个网络逼近一个函数;
- 模式识别:用一个特定的输出向量将它与输入向量联系起来;
- 分类:对输入向量以所定义的合适方式进行分类;
- 数据压缩:减少输出向量维数以便于传输或存储。

在人工神经网络的实际应用中,80%~90%的人工神经网络模型采用 BP 网络或它的变化形式,它也是前向网络的核心部分,体现了人工神经网络最精华的部分。在人们掌握反向传播网络的设计之前,感知器和自适应线性元件都只能适用于对单层网络模型的训练,只是后来才得到了进一步拓展。

虽然 BP 网络得到了广泛的应用,但自身也存在一些缺陷和不足,主要包括以下几个方面的问题。

首先,由于学习速率是固定的,因此网络的收敛速度慢,需要较长的训练时间。对于一些复杂问题,BP 算法需要的训练时间可能非常长,这主要是由于学习速率太小造成的,可采用变化的学习速率或自适应的学习速率加以改进。

其次,BP 算法可以使权值收敛到某个值,但并不保证其为误差平面的全局最小值,这是因为采用梯度下降法可能产生一个局部最小值。对于这个问题,可以采用附加动量法来解决。

再次,网络隐含层的层数和单元数的选择尚无理论上的指导,一般是根据经验或者通过反复实验确定。因此,网络往往存在很大的冗余性,在一定程度上也增加了网络学习的负担。

最后,网络的学习和记忆具有不稳定性。也就是说,如果增加了学习样本,训练好的网络就需要从头开始训练,对于以前的权值和阈值是没有记忆的。但是可以将预测、分类或聚类做得比较好的权值保存。

在 MATLAB 神经网络工具箱中提供了大量的与 BP 神经网络相关的函数。在 MATLAB



工作空间的命令行中输入“help backprop”，便可得到与 BP 神经网络相关的信息，进一步利用 help 命令又可得到相关函数的详细介绍。下面分别对这些 BP 神经网络函数进行介绍。

20.1 创建函数

在 MATLAB 神经网络工具箱中提供了 newcf 及 newff 函数用于创建 BP 神经网络。下面分别对这两个函数进行介绍。

1. newcf 函数

该函数用于创建级联前向 BP 网络函数，其调用格式为：

```
net=newcf
net=newcf(PR, [S1,S2...SN],{TF1 TF2...TFN1},BTF, BLF,PF)
```

net=newcf 用于在对话框中创建一个 BP 网络。

其中，

PR：由每组输入（共有 R 组输入）元素的最大值和最小值组成的 $R \times 2$ 维的矩阵；

Si：第 i 层的长度，共计 N1 层；

TFi：第 i 层的传递函数，默认为“tansig”；

BTF：BP 网络的训练函数，默认为“trainlm”；

BLF：权值和阈值的 BP 学习算法，默认为“learnqdm”；

PF：网络的性能函数，默认为“mse”。

【例 20-1】 假设输入和目标向量矩阵分别为

```
P=[8 7 6 5 4 3 2 1 0];
T=[0 1 2 3 2 1 2 3 2];
```

建立一个两层前向级联网络，其中第一层有六个神经元，采用 tansig 传递函数，输出层神经元的传递函数为 purelin，其他参数均采用默认值。

```
>>clear all;
P=[8 7 6 5 4 3 2 1 0];
T=[0 1 2 3 2 1 2 3 2];
net=newcf([0 8],[6 1],{'tansig' 'purelin'});
%对网络进行训练和仿真
net=train(net,P,T);
Y=sim(net,P)
```

运行程序，输出如下，效果如图 20-1 所示。

```
Y =
    0.0000    1.0000    2.0000    3.0000    2.0000    1.0000    2.0000    3.0000    2.0000
```

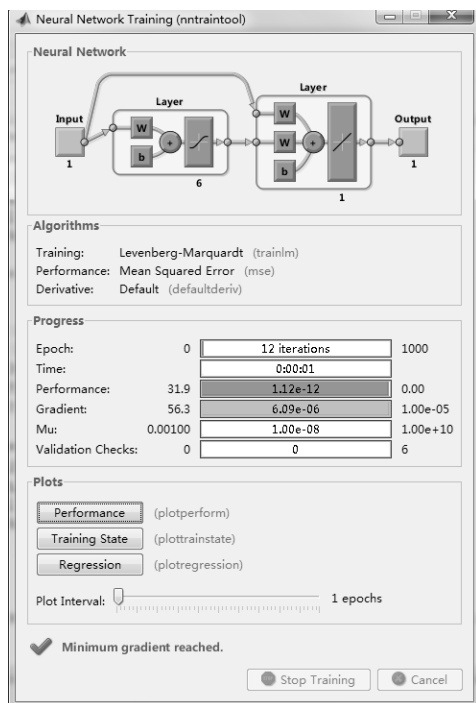


图 20-1 BP 网络训练过程

2. newff 函数

该函数用于创建一个前馈 BP 神经网络，其调用格式为：

```
net=newff(P,T,S,TF,BTF,BLF,PF,IPF,OPF,DDF)
```

其中，

P: 输入数据矩阵；

T: 目标数据矩阵；

S: 隐含层节点数；

TF: 节点传递函数，包括硬限幅传递函数 `hardlim`，对称硬幅传递函数 `hardlims`，线性传递函数 `purelin`，正切 S 型传递函数 `tansig`，对数 S 型传递函数 `logsig`。

BTF: 训练函数，包括梯度下降 BP 算法训练函数 `traingd`，动量反传的梯度下降 BP 算法训练函数 `traingdm`，动态自适应学习率的梯度下降 BP 算法训练函数 `traingda`，动量反传和动态自适应学习率的梯度下降 BP 算法训练函数 `traingdx`，Levenberg-Marquardt 的 BP 算法训练函数 `trainlm`。

BLF: 网络学习函数，包括 BP 学习规则 `learngd`，带动量项的 BP 学习规则 `learngdm`。

PF: 性能分析函数，包括均值绝对误差性能分析函数 `mae`，均方差性能分析函数 `mse`。

IPF: 输入处理函数。

OPF: 输出处理函数；

DDF: 验证数据划分函数。

一般在使用过程中设置前 6 个参数，后面 4 个参数采用系统默认参数。



【例 20-2】 利用 MATLAB 自带的 housing 数据创建一个前馈 BP 网络。

```
>> clear all;
load housing
net = newff(p,t,20);
[net,tr] = train(net,p,t);
plottrainstate(tr);
```

运行程序，效果如图 20-2 所示。

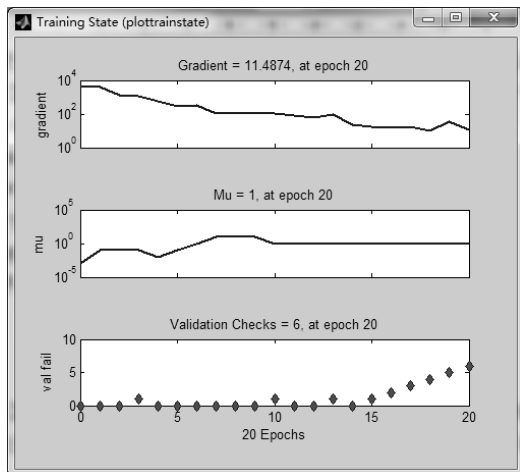


图 20-2 数据的训练状态效果图

20.2 传递函数

传递函数是 BP 网络的重要组成部分。传递函数又称为激活函数，必须是连续可微的。BP 网络经常采用 S 型的对数或正切函数和线性函数。

1. logsig 函数

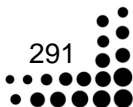
该传递函数为 S 型的对数函数，其调用格式为：

$A = \text{logsig}(N, FP)$: N 为 $S \times Q$ 维的网络输入（列）向量矩阵， FP 为性能参数（可忽略），返回网络输入向量 N 的输出矩阵 A 。

$dA_dN = \text{logsig}('dn', N, A, FP)$: 返回 A 关于 N 的导数 dA_dN ，如果 A 或 FP 没有给出或为空矩阵，则 FP 返回默认参数。

$\text{info} = \text{logsig}('code')$: 依据 $code$ 值的不同，返回不同的信息，包括：

- 当 $code=name$ 时表示返回传输函数的全称。
- 当 $code=output$ 时表示返回输出值域。
- 当 $code=active$ 返回有效的输入区间。
- 当 $code=fullderiv$ 时返回导数的次数。
- 当 $code=fpnames$ 时返回函数参数的名称。





● 当 `code=fpdefaults` 时返回默认的函数参数。

【例 20-3】 绘制一个对数 S 型传递函数曲线。

```
>> clear all;
n = -5:0.1:5;
a = logsig(n);
plot(n,a); grid on
```

运行程序，效果如图 20-3 所示。

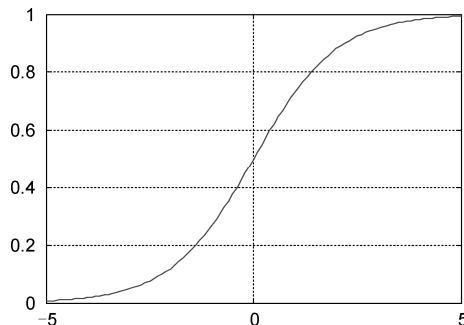


图 20-3 对数 S 型曲线

由图 20-3 可见，函数 `logsig` 可将神经元的输入（范围为整个实数集）映射到区间 $(0,1)$ ，又由于该函数为可微函数，因此非常适合于利用 BP 算法训练神经网络。

2. `tansig` 函数

该传递函数为双曲正切函数。它把神经元的输入范围 $(-\infty, +\infty)$ 映射到 $(-1, +1)$ ，并且它是可导函数，适用于 BP 训练的神经元。该函数的调用格式如下所述。

A = tansig(N,FP): N 为 $S \times q$ 维的网络输入（列）向量矩阵，FP 为性能参数（可忽略），返回网络输入向量 N 的输出矩阵 A。

dA_dN = tansig('dn',N,A,FP): 返回 A 关于 N 的导数 dA_dN，如果 A 或 FP 没有给出或为空矩阵，则 FP 返回默认参数。

【例 20-4】 一个具有双曲正切 S 型激活函数的单层网络，输入向量有 4 组，每组 3 个分量；输出向量有 5 个神经元。假定输入向量和权向量均取值为 1，试用 MATLAB 计算网络的输出。

其实现的 MATLAB 程序代码如下：

```
>> clear all;
q=4;                % 列数
r=3;                % 行数
S=5;                % 神经元数
W=ones(S,r);        % 将数 1 赋予 S×r 维权矩阵 W
B=ones(S,q);        % 将数 1 赋予 S×q 维权矩阵 B
P=ones(r,q);        % 将数 1 赋予 r×q 维权矩阵 WP
n=W*P+B;            % 计算加权输入和
A=tansig(n)          % 计算网络输出
```



运行程序，输出如下：

```
A =
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
    0.9993    0.9993    0.9993    0.9993
```

20.3 学习函数

在 MATLAB 神经网络工具箱中提供了若干函数用于实现 BP 网络的学习。

1. learngd 函数

该函数用于实现 BP 梯度下降学习。它通过神经元的输入和误差，以及权值和阈值的学习速率，来计算权值或阈值的变化率，其调用格式为：

```
[dW,LS] = learngd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
```

其中，

W：权值矩阵；

P：层输入向量；

Z：层输入经过加权函数变换后的加权输入向量；

N：加权输入经过输入函数计算后得到的神经元传递函数的输入向量；

A：该层神经元输出向量；

T：目标向量；

E：误差向量；

gW：网络性能对于权值的梯度向量；

gA：网络性能对于该层输出梯度向量；

D：神经元的距离矩阵；

LP：学习参数，learngd 函数的学习参数是由学习速率 LP.lr 构成的，默认值为 0.01；

LS：学习状态，初始值为[]。函数返回阈值调整量 dW 和当前学习状态 LS。函数的其余两种调用形式参数函数 learncon。

info = learngd('code')：根据不同的 code 值返回有关函数的不同信息，包括如下情况。

- 当 code=pnames 时返回设置的学习参数。
- 当 code=pdefaults 时返回默认的学习参数。
- 当 code=needg 时，如果函数使用了解 gW 或 gA，则返回 1。

【例 20-5】 根据给出随机梯度及速率计算权值与阈值的变化率。

```
>> clear all;
gW = rand(3,2);
```



```
lp.lr = 0.5;
dW = learngd([],[],[],[],[],[],gW,[],[],lp,[])
```

运行程序，输出如下：

```
dW =
    0.2908    0.4899
    0.0581    0.1424
    0.0288    0.2975
```

2. learnngdm 函数

该函数为梯度下降动量函数，它利用神经元的输入和误差、权值或阈值的学习速率和动量常数来计算权值或阈值的变化率，其调用格式为：

```
[dW,LS] = learnngdm(W,P,Z,N,A,T,E,gW,gA,D,LPLS)
info = learnngdm('code')
```

该函数各参数的含义与 learngd 函数相同。

【例 20-6】 根据给定的梯度、速率及动量计算其权值及阈值变化率训练网络。

```
>> clear all;
gW = rand(3,2);
lp.lr = 0.5;
lp.mc = 0.8;
ls = [];
[dW,ls] = learnngdm([],[],[],[],[],[],gW,[],[],lp,ls)
```

运行程序，输出如下：

```
dW =
    0.4811    0.1708
    0.0929    0.4664
    0.0965    0.1953

ls =
    dw: [3x2 double]
```

20.4 训练函数

在 MATLAB 神经网络工具箱中，提供了若干函数用于实现 BP 网络的训练。下面分别给予介绍。

1. trainbfg 函数

该函数为 BFGS 准牛顿 BP 算法函数。除了 BP 网络外，该函数也可以训练任意形式的神经网络，只要它的传递函数对于权值和输入存在导数即可，其调用格式为：

```
[net,TR] = trainbfg(Net,Tr,trainV,valV,testV)
```



其中,

Net: 待训练的神经网络;

Tr: 有延迟的输入向量;

trainV: 训练向量;

valV: 验证向量;

testV: 测试向量;

net: 训练后的神经网络;

TR: 每步训练的有关信息记录, 包括以下几项。

- TR.epoch: 时刻点;
- TR.perf: 训练性能;
- TR.vperf: 确认性能;
- TR.tperf: 检验性能;

info = trainbfg('info'): 返回有关函数的有用信息。

trainbfg 函数及其他一些采用准牛顿算法和共轭梯度算法的训练函数都要使用线搜索函数, 因此当网络的训练函数 net.trainFcn 设为此类函数时, 网络的训练参数将由常规参数和线搜索参数组成, 其中常规参数包括:

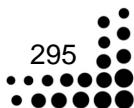
- net.trainParam.epochs: 训练次数, 默认值为 100;
- net.trainParam.goal: 网络性能目标, 默认值为 0;
- net.trainParam.max_fail: 最大验证失败次数, 默认值为 5;
- net.trainParam.min_grad: 性能函数的最小梯度, 默认值为 $1e-6$;
- net.trainParam.show: 两次显示之间的训练次数, 默认值为 25;
- net.trainParam.time: 最长训练时间 (以秒计), 默认值为 inf。

线搜索参数包括以下几项。

- net.trainParam.searchFcn: 训练时采用的线搜索方法, 默认值为 'srchcha';
- net.trainParam.scal_tol: 确定线搜索误差的尺度因子, 默认值为 20, 初始步长 delta 除以该参数后得到线搜索结果的误差;
- net.trainParam.alpha: 确定性能函数下降是否足够大的尺度因子, 默认值为 0.001;
- net.trainParam.beta: 确定步长是否足够大的尺度因子, 默认值为 0.1;
- net.trainParam.delta: 区间定位时的初始步长, 默认值为 0.01;
- net.trainParam.gama: 避免性能函数下降过小的参数, 默认值为 0.1;
- net.trainParam.low_lim: 步长变化下界, 默认值为 0.1;
- net.trainParam.up_lim: 步长变化上界, 默认值为 0.5;
- net.trainParam.maxstep: 最大步长, 默认值为 100;
- net.trainParam.minstep: 最小步长, 默认值为 $1.0e-6$;
- net.trainParam.bmax: 最大步长 (与 maxstep 在不同搜索算法中分别使用), 默认值为 26。

网络性能设置:

newff、newcf 和 newelm 函数建立的网络都可以采用 trainbfg 函数作为训练函数。只要网络的加权函数、输入函数和传递函数是可微的, 那么该网络就可以利用 trainbfg 函数进行训练,



如果要使网络利用该函数进行训练，可作如下设置：

- (1) 将网络训练函数 `net.trainFcn` 设置为 `'trainbfg'`;
- (2) 设置网络训练函数的参数 `net.trainParam`。

【例 20-7】 对所创建的 BP 网络进行训练。

```
>> clear all;
P=[0 1 2 3 4 5];
T=[0 0 0 1 1 1];
net=newff(P,T,2,{'trainbfg'}); %创建 BP 网络
disp('未训练前的 BP 网络仿真为: ')
a1=sim(net,P)
net=train(net,P,T);
disp('未训练后的 BP 网络仿真为: ')
a2=sim(net,P)
```

运行程序，输出如下，过程如图 20-4 所示。

未训练前的 BP 网络仿真为：

```
a1 =
    0.0282    0.1294    0.1509    0.1555    0.1733    0.2564
```

未训练后的 BP 网络仿真为：

```
a2 =
    0.0282    0.1294    0.1509    0.1555    0.1733    0.2564
```

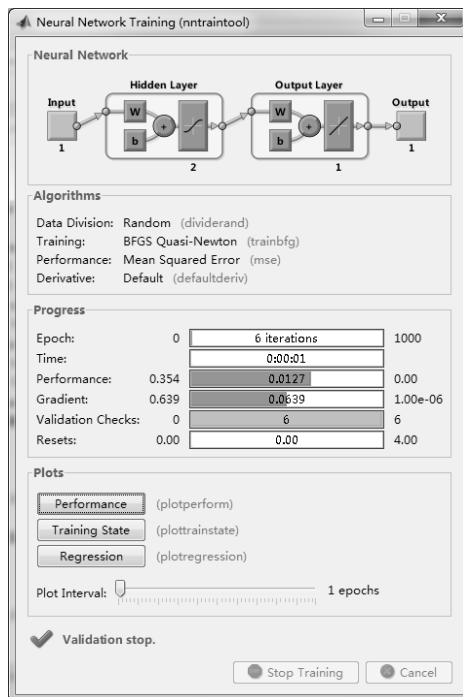


图 20-4 BP 网络的训练过程



2. traingd 函数

该函数为梯度下降 BP 算法函数，其调用格式为：

```
[net,TR] = traingd(net,TR,trainV,valV,testV)
info = traingd('info')
```

其参数意义、设置格式与适用范围等与 trainbfg 类似。

3. traingdm 函数

该函数为梯度下降动量 BP 算法函数，其调用格式为：

```
[net,TR] = traingdm(net,TR,trainV,valV,testV)
info = traingdm('info')
```

其参数意义、设置格式与适用范围等与 trainbfg 类似。

4. traingdx 函数

该函数利用快速 BP 算法训练前向网络，其调用格式为：

```
[net,TR] = traingdx(net,TR,trainV,valV,testV)
info = traingdx('info')
```

其参数意义、设置格式与适用范围等与 trainbfg 类似。

20.5 性能函数

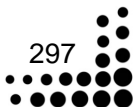
在 MATLAB 神经网络工具箱中提供了 mse 函数，用于实现 BP 网络的均方误差性能，其调用格式为：

```
perf = mse(E,Y,X,FP)
dPerf_dy = mse('dy',E,Y,X,perf,FP)
dPerf_dx = mse('dx',E,Y,X,perf,FP)
info = mse('code')
```

各参数含义参见第 3 章的 mae 函数。

【例 20-8】 创建一个 BP 网络，并评估其性能。

```
>> % 创建一个 BP 网络
net = newff([-10 10],[4 1],{'tansig','purelin'});
p = [-10 -5 0 5 10];
t = [0 0 1 1 1];
% 网络仿真
y = sim(net,p)
% 误差向量
e = t-y
```





```
%求 BP 网络的均方误差
```

```
perf = mse(e)
```

运行程序，输出如下：

```
y =
```

```
    -1.4370    -1.8532    -1.2141     0.2706     1.1952
```

```
e =
```

```
     1.4370     1.8532     2.2141     0.7294    -0.1952
```

```
perf =
```

```
     2.1944
```

20.6 显示函数

在 MATLAB 神经网络工具箱中提供了若干函数用于实现 BP 网络的显示，下面分别给予介绍。

1. plotperf 函数

该函数用于绘制一个单独神经元的误差曲面，其调用格式为：

```
plotes(WV,BV,ES,V)
```

其中，

WV 为权值的 N 维行向量；

BV 为 M 维的阈值行向量；

ES 为误差向量组成的 $M \times N$ 维矩阵；

V 为视角，默认为 $[-37.5, 30]$ 。

函数绘制的误差曲线图是由权值和阈值确定的、由函数 `errsurf` 计算得出的。

2. errsurf 函数

该函数用于计算单个神经元的误差曲面，其格式为：

```
errsurf(P,T,WV,BV,F)
```

其中，

P: 输入行向量；

T: 目标行向量；

WV: 权值列向量；

BV: 阈值列向量；

F: 传递函数的名称。

神经元的误差曲面是由权值和阈值的行向量确定的。

【例 20-9】 创建一个 BP 神经网络，分析其误差，并绘制出其误差曲面与等高线图。



```
>> % 创建一个 BP 网络  
p = [3 2];  
t = [0.4 0.8];  
% 确定误差向量及视角  
wv = -4:0.4:4; bv = wv;  
% 网络的神经元误差曲面  
ES = errsurf(p,t,wv,bv,'logsig');  
plotes(wv,bv,ES,[60 30])
```

运行程序，效果如图 20-5 所示。

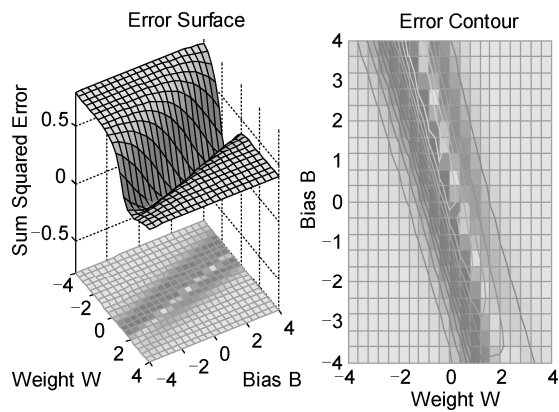


图 20-5 误差曲面与等高图

第 21 章 神经网络在实际案例中的应用



21.1 农作物虫情预测

农作物的主要害虫常年对作物造成严重危害，使农业经济遭受严重损失。根据害虫的发生、发展规律，以及作物的物候和气象预报等资料，进行全面分析，做出其未来的发生期、发生量和危害程度等估计，预测害虫的未来发展动态，这项工作称为农作物虫情预测。虫情预测工作是进行害虫综合防治的必要前提，只有对害虫发生危害的预测做得及时准确，才可以正确地拟定综合防治计划，及时采取必要的措施，经济有效地压低害虫的发生数量，保证农作物的高产和稳产。

按照预测内容来划分，虫情预测可以分为发生期预测、发生量预测、迁飞害虫预测和灾害程度预测，以及操作估计等 4 种；按照预测时间长短划分，可以分为短期预测、中期预测和长期预测等 3 种情况。

我国关于虫情预测问题的研究起步较早，目前主要采用以下 3 种方法。

- 统计法。根据多年的积累资料，探讨某种因素，如气候因素和物候现象等与害虫某一虫态的发生期和发生量之间的关系，或害虫种群本身前后不同的发生期和发生量之间的相关关系，进行相关回归分析或数理统计计算，构建各种不同的预测模型。
- 实验法。应用生物学方法，主要求出各虫态的发育速率和有效积温，然后应用气象资料预测其发生期。另一方面，利用实验方法探讨营养、气候和天敌等因素对害虫生存和繁殖能力的影响，为害虫发生量的预测提供依据。
- 观察法。直接观察害虫的发生和作物物候的变化，明确虫口密度、生活史与作物生育期的关系。应用物候现象、发育进程、虫口密度和虫态历期等观察资料进行预测，是我国目前最通用的预测方法。该方法主要可以预测发生期、发生量和灾害程度。

21.1.1 虫情预测原理

众所周知，虫害的发生和自然因素之间有着密切的联系，它同时受气温、日照和降雨量等因素的影响。影响虫害发生量的各因子之间存在复杂的相互作用。由于自身的缺点，利用传统方法很难建立起一个精确和完善的预测模型。而 BP 神经网络具有对非线性复杂系统预测的良好特性，可以有效地描述其本身具有的不确定、多输入等复杂的非线性特性。

从影响虫害发生量的气候因子角度来说，虫害发生量主要受以下 3 种因素制约。

- 温度。昆虫是变温动物，体温基本上随着外界温度的变化而变化的。而外界温度的



变化直接影响着昆虫代谢率的高低,从而直接影响昆虫生长发育、繁殖和生存等生命活动行为。昆虫对外界温度的变化适应不是无限的,而是有一定的适应范围。每种昆虫都有一定的温度适应范围,超过这一温度范围,昆虫的繁殖就会停止甚至死亡。了解每种害虫对温度的适应范围,对于分析和预测害虫的发生期和发生量有着重要的意义。

- 湿度和降雨。湿度和降雨可以直接影响昆虫的生长发育和生存。外界环境的湿度和降雨都是通过影响昆虫体内的含水量而产生作用的。所以,当外界环境的影响使得虫体内的水分调节失去平衡时,便可引起昆虫生长发育和繁殖等方面的反常表现。
- 光。对于昆虫来说,光并不是一种生存条件,但外界光因素与昆虫的趋向性、活动行为和生活方式等都有着直接或间接的联系。

自然界中的各个气候因素是相互影响、综合作用于昆虫的。在进行虫情预测时,不能够只根据某一单项指标,而是要注意综合作用的影响。

21.1.2 网络实现

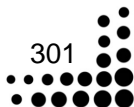
本实例的预测对象是我国某地的田间水稻。水稻螟虫是水稻的重要害虫之一,尤其是二化螟。从温度上说,二化螟的发生发展和温度的变化关系十分密切,二化螟的抗低温能力较强,抗高温能力较弱,适宜温度在 $16\sim 30^{\circ}\text{C}$ 之间, 35°C 以上的高温就容易使二化螟死亡。从降雨量角度讲,一方面,大量的降雨会导致气温下降,有利于二化螟的生存;另一方面,充沛的降雨会淹死大量的幼虫。因此,降雨对二化螟的影响是比较复杂的,需要综合考虑。

BP 网络输入和输出层的神经元数目是由输入和输出向量的维数确定的。输入向量的维数也就是影响因素的个数,这里综合考虑了影响虫情的各种因素,选取了平均气温、最低气温、日照时间和降雨量等 4 个因素,所以输入层的神经元个数为 4。为了细化虫害的等级,这里将虫害发生量分为 4 级,目标输出模式为 (0001)、(0010)、(0100) 和 (1000),分别对应 1 级、2 级、3 级和 4 级。因此,输出层神经元的个数也为 4。由于输出向量的元素为 0~1 值,因此,输出层神经元的传递函数为可选用 S 型对数函数 logsig 。

实践表明,隐含层数目的增加可以提高 BP 网络的非线性映射能力,但是隐含层数目超过一定值,网络性能反而会降低。而单隐层的 BP 网络可以逼近一个任意的连续非线性函数。因此,这里采用单隐层的 BP 网络。隐含层的神经元个数直接影响网络的非线性预测性能。这里根据 Kolmogorve 定理,设定网络的隐含层神经元个数为 9。按照一般的设计原则,隐含层神经元的传递函数为 S 型正切函数 tansig 。

网络结构确定后,需要利用样本数据通过一定的学习规则进行训练,提高网络的适应能力。学习速率是训练过程的重要因子,它决定每一次循环中的权值变化量。在一般情况下,倾向于选择较小的学习速率保证学习的稳定性,这里取学习速率为 0.05。

本实例所使用的数据为该地区的田间水稻 1996—2003 年间的 5 月到 10 月的虫害发生程度及相应的气象数据。本来应该取 1996—2002 年之间的数据作为网络的学习训练样本,2003 年的数据作为预测样本,但这里由于篇幅所限,在利用神经网络工具箱进行编程时,只利用 2000—2002 年的数据作为训练样本,2003 年的数据作为预测样本。这样做的直接后果会导致





网络预测精度下降,但这里我们更关心的是演示利用神经网络工具箱进行虫情预测的全过程。样本数据如表 21-1 所列。

表 21-1 归一化后的样本数据

年份	月	平均气温	最低气温	日照时间	降雨量	虫害程度
2000	5	-0.0909	-0.1408	-0.2500	-0.2984	0 0 0 1
	6	0.4825	0.3844	0.1250	0.3037	0 0 0 1
	7	0.9580	0.9718	0.9688	-0.7801	1 0 0 0
	8	0.6643	0.7183	0.5000	0.0419	1 0 0 0
	9	0.0350	0.0423	0.0000	-0.3665	0 0 0 1
	10	-0.6224	-0.6620	-0.0625	-0.8796	0 0 0 1
2001	5	-0.2727	-0.7324	0.5625	-0.7277	0 0 0 1
	6	-0.909	0.0000	0.8125	-0.6073	0 1 0 0
	7	0.9580	1.0000	0.6875	0.0733	1 0 0 0
	8	0.8601	0.9296	0.2812	-0.3979	1 0 0 0
	9	0.0909	0.0141	0.1563	-0.4660	1 0 0 0
	10	-0.9860	-1.0000	-0.5625	-0.4241	0 0 0 1
2002	5	0.1189	-0.1127	0.6250	-0.6021	0 0 0 1
	6	0.3706	0.3521	0.0313	-0.6073	0 0 0 1
	7	0.6923	0.7324	-0.3125	0.2670	0 0 1 0
	8	0.6643	0.7324	-0.0625	0.1361	1 0 0 0
	9	-0.0350	0.0423	-0.5313	-0.8482	1 0 0 0
	10	-0.4266	-0.5070	-0.125	-0.8586	0 1 0 0
2003	5	0.0490	0.0000	-0.0937	-0.0995	0 0 0 1
	6	0.2587	0.3662	-0.5313	1.0000	0 0 1 0
	7	0.7203	0.8028	-0.1875	0.4346	0 0 1 0
	8	0.9301	0.9014	0.9688	-0.8691	1 0 0 0
	9	0.3287	0.3239	0.2813	-0.6702	0 0 0 1
	10	-0.6084	-0.5211	-0.2813	-0.4346	0 0 0 1

该实例完整的 MATLAB 代码为:

```
%构建训练样本中的输入向量 P
p1=[-0.0909 0.4825 0.9580 0.6643 0.0350 -0.6224;
    -0.1408 0.3844 0.9718 0.7183 0.0423 -0.6620;
    -0.2500 0.1250 0.9688 0.5000 0.0000 -0.0625;
    -0.2984 0.3037 -0.7801 0.0419 -0.3665 -0.8796];
p2=[-0.2727 -0.909 0.9580 0.8601 0.0909 -0.9860;
    -0.7324 0.0000 1.0000 0.9296 0.0141 -1.0000;
    0.5625 0.8125 0.6875 0.2812 0.1563 -0.5625;
    -0.7277 -0.6073 0.0733 -0.3979 -0.4660 -0.4241];
p3=[0.1189 0.3706 0.6923 0.6643 -0.0350 -0.4266;
    -0.1127 0.3521 0.7324 0.7324 0.0423 -0.5070;
```



```

0.6250 0.0313 -0.3125 -0.0625 -0.5313 -0.125;
-0.6021 -0.6073 0.2670 0.1361 -0.8482 -0.8586];
P=[p1 p2 p3];
%构建训练样本中的目标向量 t
t1=[0 0 1 1 0 0
    0 0 0 0 0 0;
    0 0 0 0 0 0;
    1 1 0 0 1 1];
t2=[0 0 1 1 1 0;
    0 1 0 0 0 0;
    0 0 0 0 0 0;
    1 0 0 0 0 1];
t3=[0 0 0 1 1 0;
    0 0 0 0 0 1;
    0 0 1 0 0 0;
    1 1 0 0 0 0];
t=[t1 t2 t3];
%创建一个 BP 网络，隐含层有 9 个神经元，传递函数为 tansig
%中间层有 4 个神经元，传递函数为 logsig，训练函数为 trainlm
net=newff(minmax(P),[9,4],{'tansig','logsig'},'trainlm');
%训练步数为 50
%目标误差为 0.01
net.trainParam.epochs=50;
net.trainParam.goal=0.01;
net=train(net,P,t);
%预测 2003 年的虫情
P_test=[0.0490 0.2587 0.7203 0.9301 0.3287 -0.6084;
        0.0000 0.3662 0.8028 0.9014 0.3239 -0.5211;
        -0.0937 -0.5313 -0.1875 0.9688 0.2813 -0.2813;
        -0.0995 1.0000 0.4346 -0.8691 -0.6702 -0.4346];
y=sim(net,P_test)

```

运行以上代码，可以得到网络的训练过程如图 21-1 所示。

可见网络经过 28 次训练后即可达到误差要求，得到训练输出为：

```

y =
    0.0003    0.0003    0.0051    0.8198    0.7529    0.0000
    0.0000    0.0000    0.0000    0.0002    0.0005    0.0029
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.9998

```

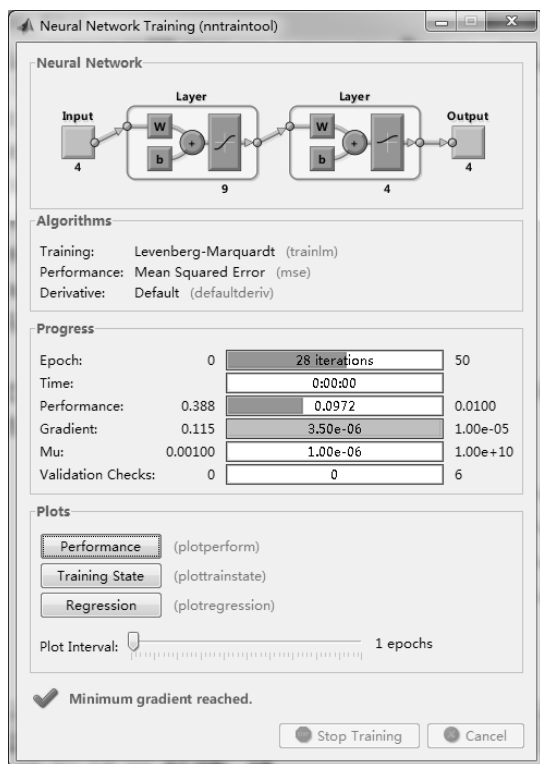


图 21-1 网络的训练过程

21.2 人脸识别

人脸识别作为一个复杂的模式识别问题，近年来受到了广泛的关注，识别领域的各种方法在这个问题上各显所长，而且发展了许多新方法，大大丰富和拓宽了模式识别的方向。人脸识别、检测、跟踪、特征定位等技术近年来一直是研究的热点。人脸识别是人脸应用研究中重要的第一步，目的是从图像中分割出不包括背景的人脸区域。由于人脸形状的不规则性及光线和背景条件的多样性，现有的人脸研究算法都是在试图解决某些特定实验环境下的一些具体问题，对人脸位置和状态都有一定的要求。而在实际应用中，大量图像和视频源中人脸的位置、朝向和旋转角度都不是固定的，这就大大增加了人脸识别的难度。

在人脸识别领域的众多研究方向中，人脸朝向分析一直是一个少有人涉及的领域。在以往的研究成果中，一些研究谈及了人脸朝向问题，但其中绝大多数都是希望在人脸识别过程中去除水平旋转对识别过程的不良影响。但是，实际问题要复杂得多，人脸朝向是一个无法回避的问题。因此，对于人脸朝向的判断和识别将会是一件非常有意义的工作。



21.2.1 模型建立

1. 问题描述

现采集到一组人脸朝向不同角度时的图像，图像来自不同的 10 个人，每人 5 幅图像，人脸的朝向分别为左方、左前方、前方、右前方和右方。试创建一个 LVQ 神经网络，对任意给定的人脸图像进行朝向预测和识别。

2. 思路设计

通过观察不难发现，当人脸面朝不同方向时，眼睛在图像中的位置差别较大。因此，可以考虑将图片中描述眼睛位置的特征信息提取出来作为 LVQ 神经网络的输入，5 个朝向分别用 1, 2, 3, 4, 5 表示，作为 LVQ 神经网络的输出。通过对训练集的图像进行训练，得到具有预测功能的网络，便可以对任意给出的人脸图像进行朝向判断和识别。

3. 实现步骤

根据上述设计思路，实现步骤主要包括以下几个部分，如图 21-2 所示。

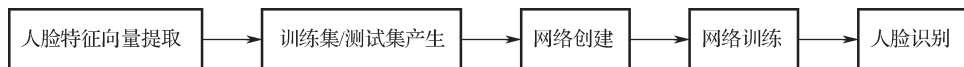


图 21-2 实现步骤流程图

(1) 人脸特征向量提取

如设计思路中所述，当人脸朝向不同时，眼睛在图像中的位置会有明显的差别。因此，只需要将描述人眼位置信息的特征向量提取出来即可。方法是整幅图像划分成 6 行 8 列，人眼的位置信息可以用第 2 行的 8 个子矩阵来描述（注意：针对不同大小的图像，划分的网格需稍做修改），边缘检测后 8 个子矩阵中的值为“1”的像素点个数与人脸朝向有直接关系，只要分别统计出第 2 行的 8 个子矩阵中的值为“1”的像素点个数即可。

(2) 训练集/测试集产生

为了保证训练集数据的随机性，随机选取图像库中的 30 幅人脸图像提取出的特征向量作为训练集数据，剩余的 20 幅人脸图像提取出来的特征向量作为测试集数据。

(3) LVQ 网络创建

LVQ 神经网络的优点是不需要将输入向量进行归一化、正交化，利用 MATLAB 自带的神经网络工具箱函数 `newlvq` 构建一个 LVQ 神经网络。

(4) LVQ 网络训练

网络创建完毕后，便可以将训练集输入向量送入网络中，利用 LVQ1 或 LVQ2 算法对网络的权值进行调整，直到满足训练要求迭代终止。

(5) 人脸识别测试

网络训练收敛后，便可以对测试集数据进行预测，即对测试集的图像进行人脸朝向识别。对于任意给出的图像，只需要将其特征向量提取出来，即可进行识别。



21.2.2 网络实现

利用 MATLAB 神经网络工具箱提供的函数可以方便地在 MATLAB 环境下实现上述步骤，其实现代码为：

1) 人脸特征向量提取

```
clear all;
%将图像中描述人眼位置的信息提取出来，即统计出划分网络第 2 行的子矩阵中的值为“1”的像素点个数
M=10;          %人数
N=5;           %人脸朝向类别数
pixel_value=feature_extraction(M,N);
```

其中，feature_extraction 函数为自定义的人脸特征提取子函数，其代码为：

```
function pixel_value=feature_extraction(m,n)
pixel_value=zeros(5,8);
sample_number=0;
for i=1:m
    for j=1:n
        str=strcat('Images\',num2str(i),'_',num2str(j),'.bmp');    %载入图像名称
        img=imread(str);
        [rows cols]=size(img);
        img_edge=edge(img,'Sobel');
        sub_rows=floor(rows/6);
        sub_cols=floor(cols/8);
        sample_number=sample_number+1;
        for subblock_i=1:8
            for ii=sub_rows+1:2*sub_rows
                for jj=(subblock_i-1)*sub_cols+1:subblock_i*sub_cols
                    pixel_value(sample_number,subblock_i)=pixel_value(sample_number, subblock_i)+img_edge(ii,jj);
                end
            end
        end
    end
end
end
```

其中，

(1) 人脸图像库的图片放在文件名为 Images 的文件夹中，图片命名规则为 i_j.bmp，其中，i 表示人的编号，j 表示人脸朝向的编号，这里，i=1,2,⋯,10，j=1,2,⋯,5。

(2) 函数 strcat 的作用是将字符串进行水平连接，具体用法可查看 MATLAB 的帮助文档。

(3) 函数 imread 用于将图片转换成对应的矩阵。



(4) edge 函数用于边缘提取，其参数 Sobel 为边缘提取算子。

2) 训练集/测试集产生

图像库中所有图片的特征向量提取出来后，随机将其分成两组，分别作为训练集和测试集。其中，训练集包含 30 个不同人脸朝向图片的特征向量，测试集为剩余的 20 个不同人脸朝向图片的特征向量。实现代码为：

```
%产生图像序号的随机序列
rand_label=randperm(M*N);
%人脸朝向标号
direction_label=repmat(1:N,1,M);
%训练集
train_label=rand_label(1:30);
p_train=pixel_value(train_label,:);
Tc_train=direction_label(train_label);
T_train=ind2vec(Tc_train);
%测试集
test_label=rand_label(31:end);
p_test=pixel_value(test_label,:);
Tc_test=direction_label(test_label);
```

其中，

- (1) 函数 randperm(n) 用于创建一个整数 1~n 的随机排列。
- (2) 函数 repmat 用于复制矩阵。

3) 创建 LVQ 网络

利用 newlvq 函数创建一个 LVQ 神经网络。此处隐含层神经元个数设置为 20。由于训练集数据是随机产生的，所以参数 PC 的设置需要事先计算得出，其实现代码为：

```
for i=1:5
    rate{i}=length(find(Tc_train==i))/30;
end
net=newlvq(minmax(p_train),20,cell2mat(rate));
%设置训练参数
net.trainParam.epochs=100;
net.trainParam.goal=0.001;
net.trainParam.lr=0.1;
```

4) 训练 LVQ 网络

网络创建完成及相关参数设置完成后，利用 MATLAB 自带的网络训练函数 train 可以方便地对网络进行训练学习，实现代码为：

```
%训练网络
net=train(net,p_train,T_train);
```



5) 人脸识别测试

利用 `sim` 函数将测试集输入数据送入训练好的神经网络, 即可得到测试集的输出仿真数据, 即测试集图像的人脸朝向识别结果, 代码如下:

```
T_sim=sim(net,p_test);  
Tc_sim=vec2ind(T_train);  
result=[Tc_test,Tc_sim];
```

其中,

`result` 第 1 行为测试集图像的标准人脸朝向类别, 第 2 行为测试集图像的预测人脸朝向类别。

6) 显示结果

实现人脸识别的显示代码如下:

```
%训练集人脸标号  
strain_label=sort(train_label);  
htrain_label=ceil(strain_label/N);  
%训练集人脸朝向类别  
dtrain_label=strain_label-floor(strain_label/N)*N;  
dtrain_label(dtrain_label==0)=N;  
%显示训练集图像序号  
disp('训练集图像为: ');  
for i=1:30  
    str_train=[num2str(htrain_label(i)) '_' num2str(dtrain_label(i)) "];  
    fprintf('%s',str_train)  
    if mod(i,5)==0  
        fprintf('\n');  
    end  
end  
%测试集人脸标号  
stest_label=sort(test_label);  
htest_label=ceil(stest_label/N);  
%测试集人脸朝向标号  
dtest_label=stest_label-floor(stest_label/N)*N;  
dtest_label(dtest_label==0)=N;  
%显示测试集图像序号  
disp('测试集图像为: ');  
for i=1:20  
    str_test=[num2str(htest_label(i)) '_' num2str(dtest_label(i)) "];  
    fprintf('%s',str_test);  
    if mod(i,5)==0  
        fprintf('\n');
```



```

end
end
%显示识别出错图像
error=Tc_sim-Tc_test;
location={'左方','左前方','前方','右前方','右方'};
for i=1:length(error)
    if error(i)~=0
        %识别出错图像人脸标号
        herror_label=ceil(test_label(i)/N);
        %识别出错图像人脸朝向标号
        derror_label=test_label(i)-floor(test_label(i)/N)*N;
        derror_label(derror_label==0)=N;
        %图像原始朝向
        standard=location(Tc_train(i));
        %图像识别结果朝向
        identify=location{Tc_test(i)};
        %图像原始朝向
        identify=location{Tc_sim(i)};
        str_err=strcat(['图像' num2str(herror_label) '_' num2str(derror_label) '识别出错.']);
        disp([str_err '(正确结果: 朝向' standard ';识别结果: 朝向' identify ')']);
    end
end
end
%显示识别率
disp(['识别率为: ' num2str(length(find(error==0))/20*100) '%']);

```

第 22 章 神经网络工具箱函数分析与应用



根据神经网络类型的不同及神经网络的发展历程，本章逐一详细地讲解 MATLAB 与神经网络工具箱函数的使用方法和主要功能。

神经网络工具箱是在 MATLAB 环境下开发出来的许多工具之一，它以人工神经网络理论为基础，利用 MATLAB 编程语言构造出许多典型神经网络的框架和相关的函数。

下面就分类型介绍相关函数的方法和功能。

22.1 神经网络构建函数的分析与应用

表 22-1 列出了神经网络的构建函数，其中 `network` 函数用于建立由用户自定义结构特殊的神经网络，这时用户要根据实际情况设定网络的属性参数。在一般情况下，用户可利用 `newp`、`newlin` 和 `newff` 等网络构建函数方便地创建各种常用网络。

表 22-1 构建函数

函数名称	功 能	函数名称	功 能
<code>network</code>	构建一个自定义神经网络对象	<code>newlin</code>	构建一个线性神经网络
<code>newc</code>	构建一个竞争层网络	<code>newlind</code>	设计一个线性神经网络
<code>newcf</code>	构建一个前向级联 BP 网络	<code>newlvq</code>	构建一个学习向量量化网络
<code>newelm</code>	构建一个 Elman 反馈网络	<code>newp</code>	构建一个感知器
<code>newff</code>	构建一个前向 BP 网络	<code>newpnn</code>	设计一个概率神经网络
<code>newfftd</code>	构建一个有输入延迟的前向 BP 网络	<code>newrb</code>	设计一个径向基函数网络
<code>newgrnn</code>	设计一个广义回归网络	<code>newrbe</code>	精确设计一个径向基函数网络
<code>newhop</code>	构建一个 Hopfield 反馈网络	<code>newsom</code>	建立一个自组织映射网络

1. `network` 函数

在 MATLAB 神经网络工具箱中，使用 `network` 函数自定义神经网络对象，其调用格式如下：

应用 建立一个自定义神经网络对象。

格式 `net=network`

`net=network(numInputs, numLayers, biasConnect, inputConnect, layerConnect, outputConnect, targetConnect)`



解析 该函数返回一个神经网络对象，函数调用形式中的 `numInputs` 等输入量为确定神经网络结构的参数，其功能说明如下。

numInputs: 该属性定义神经网络的输入个数，属性值可以是 0 或正整数。需要注意的是，该属性定义网络输入向量的总个数，而不是单个输入向量的维数。

numLayers: 该属性定义神经网络的层数，其属性值可以是 0 或正整数。

biasConnect: 该属性定义神经网络的每层是否具有阈值，其属性值为 $N \times 1$ 维的布尔量矩阵，其中 N 为网络的层数 (`net.numLayers`)。 `net.biasConnect(i)` 为 1 表示第 i 层上的神经元具有阈值，为 0 则表示该层没有阈值。

inputConnect: 该属性定义神经网络的输入层，其属性值为 $N \times N_i$ 维的布尔量矩阵，其中 N 为网络的层数， N_i 为网络的输入个数 (`net.numInputs`)。 `net.inputConnect(i,j)` 为 1，表示第 i 层上的每个神经元都要接收网络的第 j 个输入向量，为 0 则表示不接收该输入。

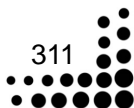
layerConnect: 该属性定义网络各层的连接情况，其属性值为 $N \times N$ 维的布尔量矩阵，其中 N 为网络的层数。 `net.layerConnect(i,j)` 为 1 表示第 i 层与第 j 层上的神经元相连，为 0 则表示它们不相连。

outputConnect: 该属性定义神经网络的输出层，其属性值为 $1 \times N$ 维的布尔量矩阵，其中 N 为网络的层数。 `net.outputConnect(i)` 为 1 表示第 i 层神经元将产生网络的输出，为 0 则表示该层不产生输出。

targetConnect: 该属性定义神经网络的目标层，即网络哪些层的输出具有目标向量。其属性值为 $1 \times N$ 维的布尔量矩阵，其中 N 为网络的层数。 `net.targetConnect(i)` 为 1 表示第 i 层神经元产生的输出具有目标向量，为 0 则表示该层输出不具有目标向量。

【例 22-1】 定义一个神经网络对象。

```
net=network
net =
    Neural Network object:
    architecture:
        numInputs: 0
        numLayers: 0
        biasConnect: []
        inputConnect: []
        layerConnect: []
        outputConnect: []
        targetConnect: []
        numOutputs: 0 (read-only)
        numTargets: 0 (read-only)
        numInputDelays: 0 (read-only)
        numLayerDelays: 0 (read-only)
    subobject structures:
        inputs: {0x1 cell} of inputs
        layers: {0x1 cell} of layers
        outputs: {1x0 cell} containing no outputs
```





```

    targets: {1x0 cell} containing no targets
    biases: {0x1 cell} containing no biases
    inputWeights: {0x0 cell} containing no input weights
    layerWeights: {0x0 cell} containing no layer weights
    functions:
        adaptFcn: (none)
        initFcn: (none)
        performFcn: (none)
        trainFcn: (none)
    parameters:
        adaptParam: (none)
        initParam: (none)
        performParam: (none)
        trainParam: (none)
    weight and bias values:
        IW: {0x0 cell} containing no input weight matrices
        LW: {0x0 cell} containing no layer weight matrices
        b: {0x1 cell} containing no bias vectors
    other:
        userdata: (user stuff)

```

上例在调用函数 `network` 时没有给出网络的结构参数，因此函数返回值 `net` 中的各属性参数均为函数提供的默认值。

2. newc 函数

应用 建立一个竞争层网络。

格式 `net=newc`

`net=newc(PR, S, KLR, CLR)`

解析 竞争层网络由单一的竞争层构成，主要用于解决分类问题。竞争层的加权函数为 `negdist`，输入函数为 `netsum`，传递函数为 `compet`。神经元权值和阈值的初始化函数分别为 `midpoint` 和 `initcon`，网络的自适应调整函数和训练函数分别为 `trains` 和 `trainr`，权值和阈值的学习函数分别为 `learnk` 和 `learncon`。其参数说明如下。

PR：为表示网络输入向量取值范围的矩阵[`Pmin Pmax`];

S：为竞争层神经元的个数，也是网络输入向量的分类数;

KLR：为权值学习速率，默认值为 0.01;

CLR：为阈值学习速率，默认值为 0.001。

【例 22-2】 利用竞争层网络把下列二维向量分为两类。

```

P=[0.1 0.8 0.1 0.9 0.2 0.8;
   0.2 0.9 0.1 0.8 0.1 0.8];

```

该二维输入向量的取值范围矩阵为

```

PR=[0 1:0 1];

```




若把这些向量分为两类，新建的竞争层应由两个神经元构成，

```
net=newc([0 1;0 1],2);
```

对网络进行训练，则语句调用格式为

```
net=train(net,P);
```

网络的仿真结果为

```
Y=sim(net,P)
```

```
Y =
```

```
(1,1)      1
(2,2)      1
(1,3)      1
(2,4)      1
(1,5)      1
(2,6)      1
```

其中，Y 是一个稀疏矩阵，其第一行的 2、4、6 列元素和第二行的 1、3、5 列元素均为 1，即 P 中第 2、4、6 个输入向量为第一类，其余三个向量属于第二类。为了便于观察结果，可将 Y 转化为下标向量形式，即

```
Yc=vec2ind(Y)
```

```
Yc =
```

```
1      2      1      2      1      2
```

3. newcf 函数

应用 构建一个前向级联 BP 网络。

格式 net=newcf

```
net=newcf(PR, [S1,S2...SN],{TF1 TF2...TFN1},BTF, BLF,PF)
```

解析 net=newcf 用于在对话框中创建一个 BP 网络，其他参数说明如下。

PR: 由每组输入（共有 R 组输入）元素的最大值和最小值组成的 $R \times 2$ 维的矩阵；

Si: 第 i 层的长度，共计 N1 层；

TFi: 第 i 层的传递函数，默认为“tansig”；

BTF: BP 网络的训练函数，默认为“trainlm”；

BLF: 权值和阈值的 BP 学习算法，默认为“learngdm”；

PF: 网络的性能函数，默认为“mse”。

【例 22-3】 假设输入和目标向量矩阵分别为

```
P=[8 7 6 5 4 3 2 1 0];
```

```
T=[0 1 2 3 2 1 2 3 2];
```

建立一个两层前向级联网络，其中第一层有六个神经元，采用 tansig 传递函数，输出层神经元的传递函数为 purelin，其他参数均采用默认值。

```
net=newcf([0 8],[6 1],{'tansig' 'purelin'});
```



对网络进行训练和仿真：

```
net=train(net,P,T);
Y=sim(net,P)
```

训练和仿真结果为

```
Y =
    0.0000    1.0000    2.0000    3.0000    2.0000    1.0000    2.0000    3.0000    2.0000
```

4. newelm 函数

应用 构建一个 Elman 反馈网络。

格式 `net=newelm`

```
net=newelm(PR, [S1,S2...SN],{TF1 TF2...TFN1},BTF, BLF,PF)
```

解析 该函数可以构建一个 N 层的 Elman 网络。网络中各层依次级联，除最后一层外网络的每层都具有自反馈，最后一层为网络的输出层。网络各层的加权函数为 `dotprod`，输入函数为 `netsum`，各层传递函数由用户设定。各神经元权值和阈值的初始化函数为 `initnw`，网络的自适应调整函数为 `trains`，并根据指定的学习函数对权值和阈值进行更新，网络的训练函数由用户指定。各参数说明见 `newcf`。

【例 22-4】 假设输入和目标序列分别为

```
Pseq={0,0,1,1,0,1,0,0,1};
Tseq={0,1,0,1,0,0,0,1,0};
```

构建一个两层 Elman 网络，其中第一层有六个神经元，传递函数为 `tansig`，该层具有自反馈，输出层神经元的传递函数为 `logsig`，其他参数均采用默认值。

```
net=newelm([0,1],[6,1],{'tansig' 'logsig'})
```

对网络进行训练和仿真：

```
net.trainParam.epochs=500;
net=train(net,Pseq,Tseq);
Y=sim(net,Pseq)
```

训练和仿真结果为

```
Y =
    0.0355    0.9606    0.0389    0.9277    0.0174    0.0301    0.0416    0.9796    0.0346
```

5. newff 函数

应用 构建一个前向 BP 网络。

格式 `net=newff(PR, [S1,S2...SN],{TF1 TF2...TFN1},BTF, BLF,PF)`

解析 在输入参数中， PR 为 R 维的输入元素的 $R \times 2$ 最大最小值矩阵； S_i 为第 i 层网络神经元的个数，共有 $N1$ 层； TF_i 为第 i 层网络的转移函数，默认为 `tansig` 函数； BTF 为神经网络的训练函数，默认为 `trainlm` 函数； BLF 为神经网络权值/偏差的学习函数； PF 为性能评价函数，默认为 `mse` 函数。

【例 22-5】 对以下样本数据，采用 `newff` 构建前向神经网络。



```
P=[8 7 6 5 4 3 2 1 0];
T=[0 1 2 3 2 1 2 3 2];
net=newff([0 8],[6 1],{'tansig' 'purelin'});
```

对网络进行训练和仿真：

```
net=train(net,P,T);
Y=sim(net,P)
```

训练和仿真结果为

```
Y =
    0.0031    0.9914    2.0086    2.9969    2.0000    0.9999    1.9998    2.9998    1.9998
```

6. newfftd 函数

应用 构建一个具有输入延迟的前向 BP 网络。

格式 `net=newfftd(PR,ID,[S1,S2...SN],{TF1 TF2...TFN1},BTF,BLF,PF)`

解析 该函数用于建立一个 N 层前向 BP 网络，网络输入经由指定延迟后进入网络的第一层。网络各层的加权函数为 `dotprod`，输入函数为 `netsum`，各层传递函数由用户设定。各神经元权值和阈值进行更新，网络的训练函数由用户指定。ID 为输入延迟向量；其他参数说明参见上面的构造函数。

【例 22-6】 假设输入和目标序列分别为

```
P={1 0 0 1 1 0 1 0};
T={1 -1 0 1 0 -1 1 -1};
```

构建一个两层网络，网络的第一层有三个神经元，该层输入由网络输入及其一拍延迟量构成，神经元传递函数为 `tansig`，输出层神经元传递函数为 `purelin`。

```
net=newfftd([0 1],[0 1],[3 1],{'tansig' 'purelin'});
```

对网络进行训练和仿真：

```
net.trainParam.epochs=60;
net=train(net,P,T);
Y=sim(net,P)
```

输出结果为：

```
Y =
    1.0000   -1.0000   -5.0460e-013    1.0000   -2.1982e-012   -1.0000    1.0000   -1.0000
```

7. newgrnn 函数

应用 设计一个广义回归网络。

格式 `net=newgrnn`

`net=newgrnn(P, T, spread)`

解析 广义回归网络是径向基函数网络的变形，主要用于解决函数逼近问题，本函数可以快速设计一个广义回归网络。广义回归网络是一个两层神经网络，第一层采用径向基神经



元，传递函数为 `radbas`，加权函数为 `dist`，输入函数为 `netprod`；第二层神经元的传递函数为 `purelin`，加权函数为 `normprod`，输入函数为 `netsum`。其参数说明如下。

P: 为网络输入样本向量构成的矩阵；

T: 为网络输出目标向量构成的矩阵；

`spread`: 为扩展常数，`spread` 越大，则函数越平滑，当 `spread` 小于输入向量间的典型距离时，网络拟合的函数将非常逼近于样本向量数据，`spread` 的默认值为 1。

【例 22-7】 假设输入和目标向量的矩阵分别为

```
P=[0 1 2 3 4];  
T=[-1 -3.1 -5.4 -4.3 -2];
```

设计一个广义回归网络，其仿真如下。

```
net=newgrnn(P,T);  
Y=sim(net,P)
```

仿真结果为

```
Y =  
    -1.8511    -3.1837    -4.3706    -3.9699    -2.8723
```

8. newlin 函数

应用 新建一个线性神经网络函数。

格式 `net=newlin`

`net=newlin(PR, S, ID, LR)`

解析 式 `net=newlin` 返回一个没有定义结构的空对象，并显示图形用户界面函数 `nntool` 的帮助文字；`net=newlin(PR, S, ID, LR)` 中 `net` 为生成的线性神经网络。其参数解析如下。

RP: 为网络输入向量中的最大值和最小值组成的矩阵 `[Pmin, Pmax]`；

S: 为输出向量的个数；

ID: 为输入延时向量（可省略）；

LR: 为学习速率（可省略），默认值为 0.01。

【例 22-8】 假设输入和目标向量矩阵为

```
P=[0 1 2 3 4];  
T=[-1 -3.1 -5.4 -7.3 -8.2];
```

建立一个单神经元线性网络。

```
net=newlin([0 5],1);
```

对网络进行训练和仿真

```
net=train(net,P,T);  
Y=sim(net,P)
```

输出结果为

```
Y =  
    -1.1557    -3.0593    -4.9629    -6.8665    -8.7701
```



9. newlind 函数

应用 设计一个线性层。

格式 `net=newlind(P,T,Pi)`

解析 其中 `P`、`T` 分别是训练样本的输入矩阵和目标输出向量，`Pi` 是初始输入延时 cell 向量。

【例 22-9】 使用 `newlind` 函数构建线性神经网络逼近层：

$$y = \begin{cases} \sin(2\pi t) & 0 \leq t \leq 2 \\ \sin(4\pi t) & 2 \leq t \leq 4 \\ \sin(8\pi t) & 4 \leq t \leq 6 \end{cases}$$

首先，产生输入训练样本和训练目标样本。

```
time1=0:0.01:2;
time2=2.01:0.01:4;
time3=4.01:0.01:6;
time=[time1 time2 time3];
T=[cos(time1*2*pi) cos(time2*4*pi) cos(time3*8*pi)];
Q=length(T);
P=zeros(6,Q);
P(1,2:Q)=T(1,1:(Q-1));
P(2,3:Q)=T(1,1:(Q-2));
P(3,4:Q)=T(1,1:(Q-3));
P(4,5:Q)=T(1,1:(Q-4));
P(5,6:Q)=T(1,1:(Q-5));
P(6,7:Q)=T(1,1:(Q-6));
```

然后使用 `newlind` 函数设计线性层。

```
net=newlind(P,T);
a=sim(net,P);
plot(time,T,time,a,'k-o');
legend('给定输入信号','网络输出信号')
figure
plot(time,a-T)
title('误差曲线')
```

线性层网络函数逼近结果如图 22-1 和 22-2 所示。

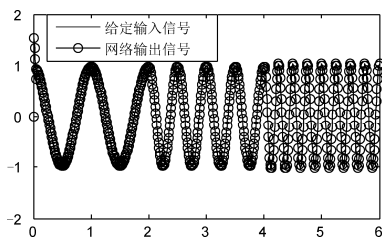


图 22-1 newlind 的函数逼近效果图

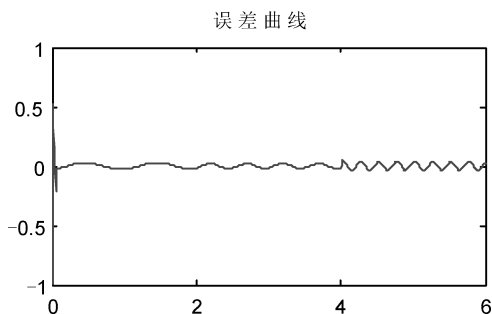


图 22-2 训练误差曲线效果图

10. newlvq 函数

应用 建立一个学习向量量化网络。

格式 `net=newlvq`

`net=newlvq(PR, S1, PC, LR, LF)`

解析 学习向量量化网络主要用于解决分类问题。该网络由两层神经元组成，网络的第一层为竞争；第二层中的每个神经元只与竞争层中的部分神经元相连，网络中的每个神经元都没有阈值。网络第一层神经元的传递函数为 `compet`，加权函数为 `negdist`，权值初始化函数为 `midpoint`；网络第二层神经元的传递函数为 `purelin`，加权函数为 `dotprod`，权值都为 1。网络使用 `train` 和 `trainr` 函数进行自适应调整和训练，这两个函数使用指定的学习函数对竞争层神经元的权值进行修正。其参数说明如下。

PR：表示网络输入向量取值范围的矩阵[Pmin, Pmax]；

S1：为竞争层神经元的个数；

PC：为 s^2 （直接由 PC 的维数确定）维向量，其中 s^2 为网络输入向量的类别数，也是输出层的神经元个数；PC(i)为网络输入向量中第 i 类向量所占的比例，即输出层中和竞争层第 i 个神经元相连的神经元所占的比例；

LP：为学习速率，默认值为 0.01；

LF：为学习函数，可以是 `learnlv1` 或 `learnlv2` 函数，默认值为 `'learnlv1'`，网络只有先使用 `learnlv1` 函数进行学习后，才能再用 `learnlv2` 进行学习。

【例 22-10】 假定要将下列样本向量分为两类，每类所占的比例均为 0.5：

```
P=[3 -2 -1 2 2 -1;
    2 -1 -1 2 3 -2];
T=[1 2 2 1 1 2];
```

建立学习向量量化网络，网络竞争层由四个神经元组成。

```
net=newlvq([-2 3; -2 3],4,[0.5 0.5]);
```

下标向量 T 必须转化为单值向量组才能作为目标向量使用：

```
T1=ind2vec(T);
net=train(net,P,T1);
```

仿真结果为



```
Y=vec2ind(sim(net,P))
```

```
Y =
```

```
    1    2    2    1    1    2
```

11. newhop 函数

应用 该函数用于设计一个 Hopfield 网络。

格式 `net=newhop`

`net=newhop(T)`

解析 `net=newhop`: 用于在对话框中创建一个 Hopfield 网络;

T: Q 个目标向量组成的 $R \times Q$ 维矩阵, 矩阵元素必须为 1 或 -1;

net: 函数返回值, 创建的 Hopfield 网络, 稳定点位于目标向量 **T** 中。

【例 22-11】 创建一个 Hopfield 网络并进行仿真。

```
T=[-1 -1 1;1 -1 1];
%创建一个 Hopfield 网络
net=newhop(T);
Ai=[[-0.9; -0.8;0.7]];
[Y,Pf,Af]=sim(net,{1 5},{},Ai);
Y{1}'
```

结果为

```
ans =
    -1    -1     1
```

由于 $Y\{1\}=T(:, 1)$, 可以看出, 设计的 Hopfield 网络已经成功地将存在偏差的向量 A_i 转换为最接近的目标向量 **T**。

12. newp 函数

应用 构造感知器模型。

格式 `net=newp(PR, S, TF, LF)`

解析 **PR**: $R \times 2$ 的输入向量最大值和最小值构成的矩阵;

S: 神经元的个数;

TF: 传输函数设置, 为 `hardlim` 函数或者 `hardlins` 函数, 默认为 `hardlim` 函数;

LF: 学习函数设置, 为 `learnp` 函数或者 `learnpn` 函数, 默认为 `learnp` 函数;

net: 生成的感知器网络。

【例 22-12】 生成具有 1 个神经元的神经网络。

```
%输入向量
p=[1.24 1.36 1.38 1.38 1.38 1.4 1.48 1.54 1.56 1.14 1.18 1.2 1.26 1.28 1.3;
    1.72 1.74 1.64 1.82 1.9 1.7 1.82 1.82 2.08 1.78 1.96 1.86 2.0 2.9 1.96];
%目标向量
T=[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0];
net=newp([0 3;0 3],1);
```



查看工作窗口中的变量，可以得到：

```
>> whos

Name      Size      Bytes  Class
T         1x15         120  double array
net       1x1       18543  network object
p         2x15         240  double array

Grand total is 676 elements using 18903 bytes
```

net 的变量属性是网络对象的结构体，可以输入 net 后查看 net 结构体的相关信息。

```
>> inputweights=net.inputweights{1}
inputweights =
    delays: 0
    initFcn: 'initzero'
    learn: 1
    learnFcn: 'learnp'
    learnParam: []
    size: [1 2]
    userdata: [1x1 struct]
    weightFcn: 'dotprod'
```

通过以下命令查看神经网络各层权重以及神经元阈值情况。

```
>> net.IW{1}      %输入层神经元权重
ans =
     0     0
>> net.LW{1}      %隐层神经元权重
ans =
     []
>> net.b{1}       %神经元阈值
ans =
     0
```

13. newrb 函数

应用 构建径向基网络。

格式 [net, tr]=newrb(P, T, goal, spread, MN, DF)

解析 在径向基网络设计中，spread 参数对径向基网络的性能影响较大，在通常情况下，spread 的值越大时，径向基网络逼近曲线越光滑，当 spread 值过小时，径向基函数的逼近效果就会变差。其参数说明如下。

P、T：分别为训练样本输入和目标输出；

goal：为径向基网络输出的总平均误差方差；

spread：为径向基函数 radbas 的密度常数；

MN：为最大的神经数目。



【例 22-13】 创建一个径向基网络。

```
P=[1 2 3 4 5];
T=[0.5 2.3 3.2 4.6 6.7];
err_goal=0.01
spr_cons=1;           %径向基函数密度常数
net=newrb(P,T,err_goal,spr_cons);
A=sim(net,P)          %回代检验
postreg(A,T)
```

径向基网络输出和训练目标输出的回归线如图 22-3 所示。回代检验的输出结果为：

```
A =
    0.5000    2.3000    3.2000    4.6000    6.7000
```

对新样本进行训练，输入以下程序段：

```
>> p=3.5;
>> a=sim(net,p)
```

输出结果为：

```
a =
    3.5678
```

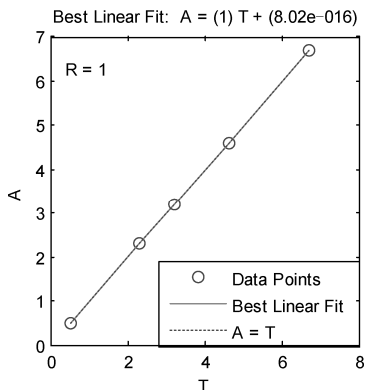


图 22-3 newrb 函数径向基网络输出回归曲线

14. newrbe 函数

应用 创建一个准确的径向基网络，可以实现网络输出与目标输出零误差。

格式 net= newrbe(P,T,spread)

newrbe 函数参数的意义同 newrb 完全相同，具体介绍见 newrb 函数的相关介绍。

【例 22-14】 创建一个准确的径向基函数。

```
P=[1 2 3 4 5];
T=[0.5 2.3 3.2 4.6 6.7];
spr_cons=1;
net=newrbe(P,T,spr_cons);
```



```
A=sim(net,P)
postreg(A,T)
p=3.5;
a=sim(net,p)
```

该程序段首先使用 `newrbe` 函数创建一个准确的径向基函数，然后使用训练样本数据回代检验，并绘制网络输出的回归曲线，最后给出测试样本的输出。`newrbe` 函数的准确径向基网络输出同目标向量的回归曲线如图 22-4 所示。可以发现此时 `newrbe` 函数创建的准确径向基网络能够实现目标向量零误差输出。回代检验结果和测试样本输出结果如下：

```
A =
    0.5000    2.3000    3.2000    4.6000    6.7000
a =
    3.6339    %同 newrb 函数创建的径向基网络输出有差异
```

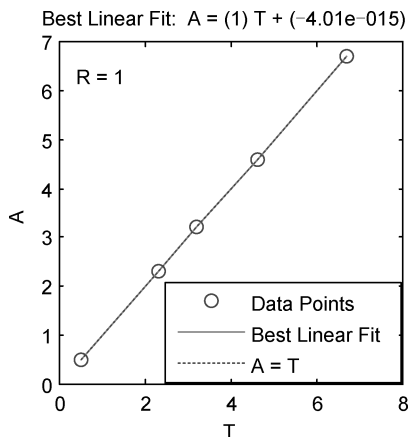


图 22-4 `newrbe` 函数的准确径向基网络输出同目标向量的回归曲线

15. newpnn 函数

应用 创建一个概率神经网络。

格式 `net=newpnn(P,T,spread)`

解析 `newpp` 函数参数意义参见 `newrb` 函数相关介绍。下面通过实例演示 `newpnn` 函数的使用。

【例 22-15】 创建一个概率神经网络。

```
P=[0 0 2 3 2 2 1;0 2 1 2 1 3 1];
T=[1 1 2 1 2 1 2];
Tc=ind2vec(T)
```

`ind2vec` 函数将下标变换成单值向量：

```
Tc =
    (1,1)    1
    (1,2)    1
```



```
(2,3)      1
(1,4)      1
(2,5)      1
(1,6)      1
(2,7)      1
```

创建概率神经网络，并对输入样本进行训练：

```
>> net=newpnn(P,Tc);      %创建概率神经网络
Ac=sim(net,P)
A=vec2ind(Ac)             %将单值向量转化为下标形式
```

输出结果为：

```
A =
     1     1     2     1     2     1     2
```

从输出结果可以看出，回代误差为 0，对输入向量进行了正确的分类。对新的样本进行训练：

```
>> p=[2 3;2 1]
a=vec2ind(sim(net,p))
```

返回结果为：

```
a =
     1     2
```

16. newsom 函数

应用 建立一个自组织映射网络。

格式 `net=newsom`

`net=newsom(PR, [D1,D2,...], TFCN, DFCN, OLR, OSTEPS, TLR, TND)`

解析 `net=newsom`：表示在对话框中创建一个新的网络；

PR：R 个输入元素的最大值和最小值设定值， $R \times 2$ 维矩阵；

DI：第 i 层的维数，默认为[5, 8]；

TFCN：拓扑函数（即结构函数），默认为 `hexstop`；

DFCN：距离函数，默认为 `linkdist`；

OLR：分类阶段学习速率，默认为 0.9；

OSTEPS：分类阶段的步长，默认为 1000；

TLR：调谐阶段的学习速率，默认为 0.02；

TND：调谐阶段的领域距离，默认为 1。

【例 22-16】 假定要把下列样本向量分为四类

```
P=[3 -2 1 2 -2 -1 -2 1;
   2 -1 -1 2 3 -2 2 3];
```

则需要建立一个四神经元的自组织映射网络。



```
net=newsom([-2 3; -2 3],[2 2]);
```

网络训练和仿真结果为

```
net=train(net,P);
Y=vec2ind(sim(net,P))
Y =
     4     1     3     4     2     1     2     4
```

22.2 神经网络应用函数的分析与应用

表 22-2 列出了神经网络的基本应用函数，它们可分别用来对神经网络进行训练、显示、初始化和仿真，调用这些函数可以完成对神经网络对象的基本操作。

表 22-2 神经网络的应用函数

函数名称	功 能	函数名称	功 能
adapt	对神经网络进行自适应调整	initlay	对神经网络逐层进行初始化
disp	显示神经网络对象的属性	revert	对神经网络的权值和阈值重新设置为初始值
display	显示神经网络对象的名称和属性		
init	对神经网络的参数进行初始化	train	对神经网络进行训练
initnw	待初始化的神经网络	sim	对神经网络进行仿真
initwb	层初始化函数		

1. adapt 函数

应用 对神经网络进行自适应训练。

格式 [net, Y, E, Pf, Af, tr]=adapt(net, P,T, Pi, Ai)

解析 输入参数说明如下。

P: 训练样本数据，对于 n 组训练样本， R 个输入端，则 P 的数据格式为 $R \times n$ 的数组。

T: 目标样本数据，相对应于 P 数据，则为 $1 \times n$ 的输出向量。

Pi: 初始化输入层延迟条件。

Ai: 初始化层延迟条件。

输出参数说明如下。

Y: 神经网络的输出数据，数据大小格式同输入的目标样本数据，为 $1 \times n$ 的向量。

E: 神经网络的输出误差数据，大小为 $1 \times n$ 的向量，使用 sse 函数或者 mse 函数计算网络误差。

Pf: 训练后的网络输出层延迟条件。

Af: 训练后网络层延迟条件。

tr: 网络训练过程数据记录，包括 epochs 和 pref 数据。

对例 22-12 构建的感知器网络进行自适应训练，得到以下结果：



```
>> [net,Y,E,Pf,Af,tr]=adapt(net,p,T);
>> Y
Y =
    1    1    1    1    1    1    1    1    1    1    1    1    1    1
>> E
E =
    0    0    0    0    0    0    0    0    0    0   -1   -1   -1   -1   -1   -1
>> tr
tr =
    timesteps: 1
          perf: 0.4000
>> mse(E)
ans =
    0.4000
>> sse(E)
ans =
    6
```

2. disp 和 display 函数

应用 显示神经网络对象的属性。

格式 `disp(net)`

`display(net)`

解析 这两个函数都用于显示神经网络对象的属性，唯一不同之处在于 `display` 函数还要显示神经网络对象的名称，而 `disp` 则不显示。

3. init 函数

应用 对神经网络的参数进行初始化。

格式 `net=init(net)`

解析 该函数通过调用初始化函数 `net.initFcn`，并根据初始化函数参数 `net.initParam` 对网络权值和阈值进行初始化。

4. initlay 函数

应用 对每一层的结构神经网络进行初始化。

格式 `NET=initlay(net)`

`info=initlay(code)`

解析 `net`: 待初始化的神经网络;

`NET`: 初始化后的神经网络;

`info=initlay(code)`: 根据不同的 `code` 代码返回不同的信息，包括

`pnames`: 初始化参数的名称;

`pdefaults`: 默认的初始化参数。

通过指定神经网络每一层 `i` 的初始化函数 `NET.layers{i}` 来调用该函数，初始化后的神经



网络每一层都得到了修正。

5. initnw 函数

应用 对层神经网络进行初始化。

格式 `NET=initnw(net, i)`

解析 `net`: 待初始化的神经网络;

`i`: 层次索引;

`NET`: 初始化后的神经网络。

6. initwb 函数

应用 该函数也是一个层初始化函数，它按照设定的每层的初始化函数对每层的权值和阈值进行初始化。

格式 `NET=initwb(net, i)`

解析 `net`: 待初始化的神经网络;

`i`: 层次索引;

`NET`: 初始化后的神经网络。

【例 22-17】 根据给定的输入向量 `P` 和目标向量 `T`，创建一个感知器网络，对其进行训练并初始化。

```
P=[0 1 0 1;0 0 1 1];
T=[1 0 0 0];
net=newp([0 1;-2 2],1); %创建一个感知器网络
%感知器的 P 和阈值
net.iw{1,1}
net.b{1}
%对感知器进行训练
net=train(net,P,T);
net.iw{1,1}
net.b{1}
net=init(net);
net.iw{1,1}
net.b{1}
```

输出结果为:

```
ans =
      0      0
ans =
      0
TRAINC, Epoch 0/100
TRAINC, Epoch 4/100
TRAINC, Performance goal met.
ans =
```



```
-1    -1
ans =
    0
ans =
    0    0
ans =
    0
```

可见，在感知器刚刚创建，尚未进行训练以前，权值和阈值都为 0；训练以后，权值和阈值分别为[1 2]和-3；对训练好的网络进行初始化后，恢复到以前的值，都为 0。

7. revert 函数

应用 该函数将网络中的权值和阈值恢复为初始值。

格式 `net=revert(net)`

解析 该函数将网络中的权值和阈值恢复为最近一次初始化时产生的初始数值，如果网络结构在初始化后发生了变化，权值和阈值将不能恢复，这时将把它们都设置为 0。

8. train 函数

应用 用于对神经网络进行训练。

格式 `[net, tr, Y, E, Pf, Af]=train(NET, P, T, Pi, Ai)`

`[net, tr, Y, E, Pf, Af]= train(NET, P, T, Pi, Ai, VV, TV)`

解析 输出参数说明如下。

NET: 待训练的神经网络；

P: 网络的输入信号；

T: 网络的目标，默认为 0；

Pi: 初始的输入延迟，默认为 0；

Ai: 初始的层次延迟，默认为 0；

VV: 网络结构确认向量，默认为空；

TV: 网络结构测试向量，默认为空。

输入参数说明如下。

net: 函数返回值，训练后的神经网络；

tr: 函数返回值，训练记录（包括步数和性能）；

Y: 函数返回值，神经网络输出信号；

E: 函数返回值，神经网络的误差；

Pf: 函数返回值，最终输入延迟；

Af: 函数返回值，最终层延迟。

【例 22-18】 使用 train 函数训练感知器网络。

首先，输入训练样本和训练目标样本，创建单神经元感知器网络，设置训练函数 train 的参数，并进行训练，训练过程曲线如图 22-5 所示。



```

%输入向量
P=[1.24 1.36 1.38 1.38 1.38 1.4 1.48 1.54 1.56 1.14 1.18 1.2 1.26 1.28 1.3;
    1.72 1.74 1.64 1.82 1.9 1.7 1.82 1.82 2.08 1.78 1.96 1.86 2.0 2.9 1.96];
%目标向量
T=[1 1 1 1 1 1 1 1 1 0 0 0 0 0 0];
net=newp([0 3;0 3],1);
%训练函数 train 的参数设置
net.trainParam.epochs=500;
net.trainParam.goal=0;
net.trainParam.show=50;
%使用 train 函数训练单神经元感知器网络
[net,tr,Y,E,Pf,Af]=train(net,P,T);
%使用测试样本数据并绘制分类图
figure
p=[1.24,1.28,1.4,;1.8,1.84,2.04];
a=sim(net,p);
plotpv(p,a);
point=findobj(gca,'type','line');
set(point,'Color','red');
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});

```

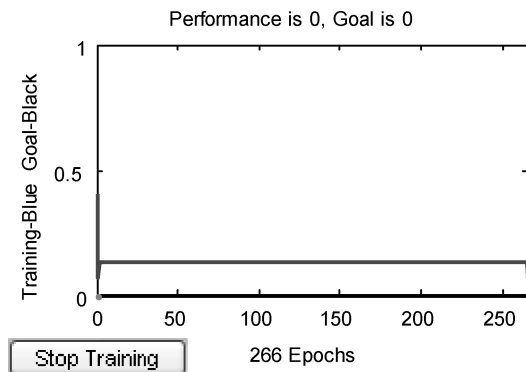


图 22-5 单神经元感知器训练过程曲线

然后利用训练好的单神经元感知器模型仿真检验样本数据，结果如图 22-6 所示。

9. sim 函数

应用 对神经网络进行仿真。

格式 [Y, Pf, Af, E, pref]=sim(net, P, Pi, Ai, T)

[Y, Pf, Af, E, pref]=sim(net, {Q TS}, Pi, Ai, T)

[Y, Pf, Af, E, pref]=sim(net, Q, Pi, Ai, T)

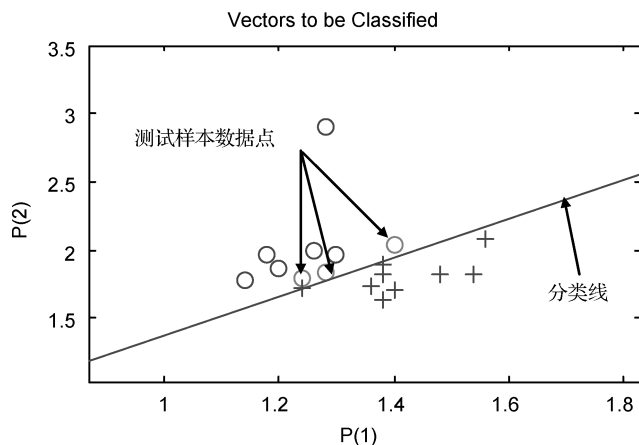


图 22-6 测试样本训练结果

解析 输入输出参数的设置和说明可以参考 `adapt` 函数中的说明。使用 `sim` 函数同样可以实现感知器网络的训练，利用前面建立的感知器神经元模型，训练新的样本数据。

```
p=[1.24,1.28,1.4;1.8,1.84,2.04];
a=sim(net,p);
```

输出结果：

```
a =
    1    1    1
```

第 23 章 线性神经网络算法分析与设计

线性神经网络是最简单的一种神经网络，它由一个或多个线性神经元构成。1960 年由 B.Widrow 和 M.E.Hoff 提出的自适应线性单元（Adaline）网络是线性神经网络最早的典型代表。线性神经网络采用线性函数作为传递函数，因此其输出可以取任意值。线性神经网络可以采用基于最小二乘算法（LMS）的 Widrow-Hoff 学习规则来调节网络的权值和阈值，其收敛速度和精度都有较大的改进。此外，采用 newlind 函数还可以直接根据网络的输入向量和目标向量设计出期望的线性网络。和感知器神经网络一样，线性神经网络只能反映输入和输出样本向量间的线性映射关系，它只能解决线性可分问题。线性神经网络在函数拟合、信号滤波、预测和控制等方面都有着广泛的应用。

23.1 线性神经网络结构

1. 线性神经元模型

线性神经元模型如图 23-1 所示。

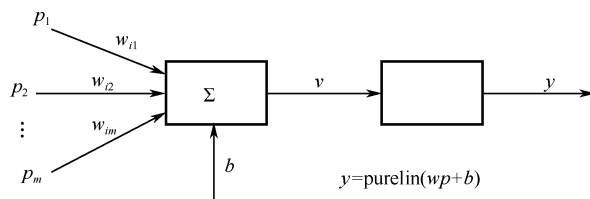


图 23-1 线性神经元模型

与感知器神经元不同的是，线性神经元采用的传递函数为线性函数 purelin，其输入与输出之间是简单的纯比例关系。purelin 函数的具体应用参见第 3 章。线性神经元的输出可以取任意值，其输入、输出关系为：

$$a = \text{purelin}(wp + b) = wp + b \quad (23-1)$$

2. 线性神经网络结构

图 23-2 给出了具有 R 维输入的单层（包含 S 个神经元）线性神经网络模型。

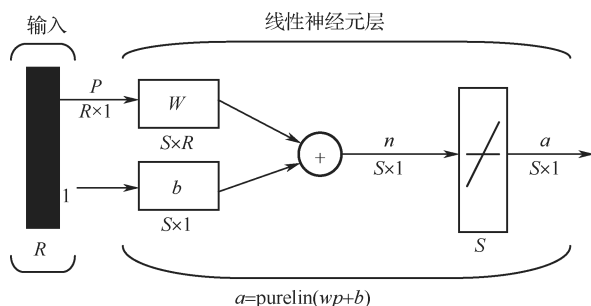


图 23-2 线性神经网络

23.2 线性神经网络设计

1. 线性神经网络的学习规则

线性神经网络权值和阈值的学习规则采用的是基于最小二乘原理的 Widrow-Hoff 学习算法。基于 Widrow-Hoff 学习算法的权值和阈值调节原理如下：

$$W(k+1) = W(k) + \Delta W(k), \quad b(k+1) = b(k) + \Delta b(k) \quad (23-2)$$

其中

$$\Delta W(k) = -\alpha \frac{\partial e^2(k)}{\partial W} \quad \Delta b(k) = -\alpha \frac{\partial e^2(k)}{\partial b} \quad (23-3)$$

由于

$$\frac{\partial e^2(k)}{\partial W} = 2e(k) \frac{\partial e}{\partial W} \quad \frac{\partial e^2(k)}{\partial b} = 2e(k) \frac{\partial e}{\partial b} \quad (23-4)$$

且有

$$\frac{\partial e}{\partial W} = \frac{\partial}{\partial W} [t(k) - (W * p(k) + b)] = -p(k) \quad \frac{\partial e}{\partial b} = -1 \quad (23-5)$$

所以，上述权值和阈值的调节公式可变为：

$$W(k+1) = W(k) + 2\alpha e(k) p^T(k) \quad b(k+1) = b(k) + 2\alpha e(k) \quad (23-6)$$

即

$$W(k+1) = W(k) + \eta e(k) p^T(k) \quad b(k+1) = b(k) + \eta e(k) \quad (23-7)$$

其中， η 为学习速率。当 η 较大时，学习速率加快（ η 取值过大时有可能使学习变得不稳定），反之亦然。

在 MATLAB 中，Widrow-Hoff 学习算法对应的学习函数为 learnwh。

2. 线性神经网络的训练和仿真

对线性神经网络的训练可以调用 train 函数完成。利用 train 函数对线性神经网络进行训练，实际上是根据所给出的“输入-目标”样本向量集，调用神经网络生成时所定义的权值和阈值学习函数 learnwh，对网络不断进行调节，最终使网络输出接近目标输出的过程。下面的代码给出了一个典型的具有二维输入线性神经元的训练过程。



```
clear all;
P=[1 2 -1 0;2 2 0 -1];
T=[0 0 1 1];
net=newlin([-2 2; -2 2],1);
net.trainParam.goal=0.01;
[net,tr]=train(net,P,T);
```

训练中会显示如下信息:

```
TRAINB, Epoch 0/100, MSE 0.5/0.01.
TRAINB, Epoch 25/100, MSE 0.121667/0.01.
TRAINB, Epoch 50/100, MSE 0.0334739/0.01.
TRAINB, Epoch 75/100, MSE 0.012111/0.01.
TRAINB, Epoch 82/100, MSE 0.00983141/0.01.
TRAINB, Performance goal met.
```

可见, 当训练到第 82 步时, 网络性能达标。此时的权值和阈值矩阵为:

```
>> W=net.iw{1,1}
W =
    -0.1854    -0.2112
>> b=net.b{1}
b =
    0.6973
```

利用 `sim` 等相关函数可以对训练好的线性神经网络进行仿真和误差分析:

```
>> a=sim(net,P)
a =
    0.0894   -0.0959    0.8827    0.9086
>> error=T-a
error =
   -0.0894    0.0959    0.1173    0.0914
>> m=mse(error)
m =
    0.0098
```

3. 线性神经网络的直接设计法

与其他神经网络设计不同的是, 线性神经网络可以根据输入和目标向量直接设计出来。函数 `newlind` 无须经过训练, 就可以直接设计出线性神经网络, 使得网络实际输出与目标输出的平方和误差 SSE 为最小, 其常用格式为:

```
net=newlind(P,T)
```



23.3 自适应滤波线性神经网络

1. 自适应滤波线性神经网络结构

自适应滤波是线性神经网络一个很重要的应用。自适应滤波线性神经网络（简称自适应滤波网络）是线性神经网络的一种特殊形式，它与一般线性神经网络的不同之处在于自适应滤波网络引入了如图 23-3 所示的延迟链（TDL）。

可见，自适应滤波网络的输入是由同一输入信号的若干延迟信号所构成的，即有

$$\begin{aligned} pd_1(k) &= p(k) \\ pd_2(k) &= p(k-1) \\ &\vdots \\ pd_N(k) &= p(k-N+1) \end{aligned}$$

一个典型的自适应滤波网络的结构如图 23-4 所示，其中，线性层只有一个神经元。该神经网络的输出为：

$$a(k) = \text{purelin}(wp + b) = \sum_{i=1}^N w_{1,i} p(k-i+1) + b \quad (23-8)$$

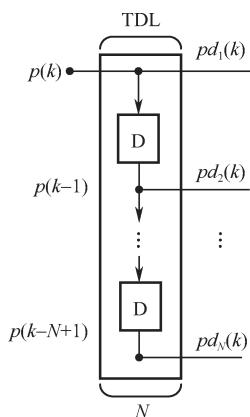


图 23-3 TDL 延迟链

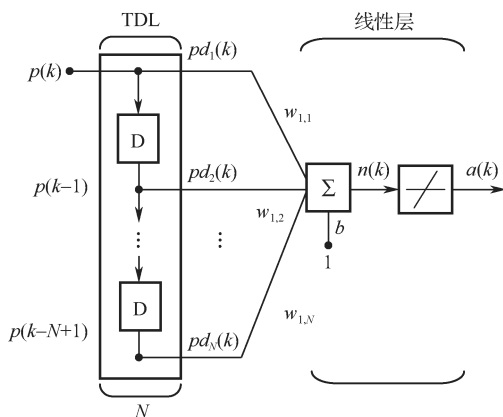


图 23-4 自适应滤波网络结构

可见，自适应滤波网络的输出是由输入信号 $p(k)$ 及其延迟信号的线性组合构成的，这正好与数字信号处理领域所谓的有限冲击响应（FIR）滤波器的结构形成相吻合。

当然，自适应滤波网络的线性层也可以由多个线性神经元组成。一个典型的具有多个神经元的自适应滤波网络结构的缩略形式如图 23-5 所示。

迄今为止，自适应滤波网络已成为应用最为广泛的神经网络之一，它在信号滤波、预测与控制等领域都有着广泛的应用。

2. 自适应滤波线性神经网络设计

自适应滤波网络的生成可以采用两种方式：一种是通过调用 `newlin` 函数直接生成带有延

迟链的自适应滤波网络；另一种则是首先利用 `newlin` 函数生成不带延迟链的线性网络，然后通过网络重定义将延迟链加入预先生成的线性网络中。图 23-6 为一个单神经元自适应滤波网络的结构示意图。

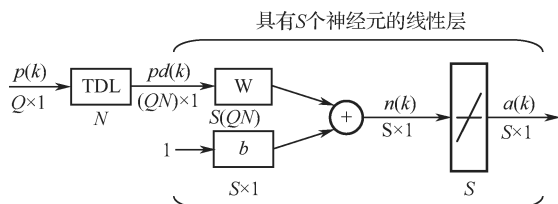


图 23-5 具有 S 个神经元的自适应滤波网络的缩略形式

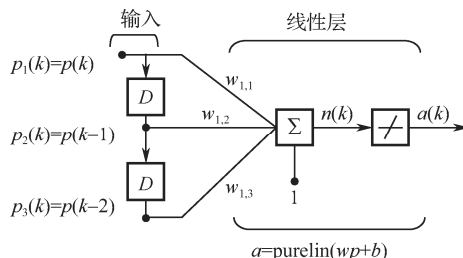


图 23-6 单神经元自适应滤波网络结构

该自适应滤波网络可以采用如下两种方式生成（假设网络输入范围为[0, 10]）：

方式一：

```
net=newlin([0, 10], 1, [0 1 2]);
```

方式二：

```
net=newlin([0, 10], 1);
net.inputWeights{1,1}.delays=[0 1 2];
```

其中，[0 1 2]中各元素分别表示自适应滤波网络各维输入所对应的延迟量。下面我们对所生成的自适应滤波网络分别进行初始化、仿真和训练。

自适应滤波网络的初始化与一般的线性神经网络基本相同，只是在初始化网络权值和阈值的同时，自适应滤波网络还要对延迟输入的初始值进行设置。

首先，对网络初始权值和阈值进行设置：

```
net=newlin([0,10],1,[0 1 2]);
net.iw{1,1}=[7 8 9];
net.b{1}=0;
```

其次，在对网络进行仿真之前需要对延迟输入 $p_2(k)$ 和 $p_3(k)$ 的初始值进行设置：

```
Pi={1,3};
```

即对应 $p_2(0) = 1$ ， $p_3(0) = 3$ 。

定义输入向量如下：

```
P={3 4 5 6};
```



现在可以调用 `sim` 函数对上面所创建的自适应滤波网络进行仿真：

```
[A,Pf]=sim(net,P,Pi)
A =
    [54]    [79]    [94]   [118]
Pf =
    [5]    [6]
```

其中，`Pf` 为仿真后的延迟输入的终值。有兴趣的读者可以利用笔算对上述网络的仿真结构加以验证，以便能更好地理解自适应滤波网络的工作原理。

下面，利用 `adapt` 函数对自适应滤波网络进行训练，使自适应滤波网络能够根据输入序列 `P` 输出期望的目标序列 `T`。

```
T={10 30 50 70};
net.adaptParam.passes=300;
[net,Y,E]=adapt(net,P,T,Pi)
```

经过训练，网络的实际输出响应为：

```
Y =
    [14.2170]    [40.5601]    [43.9006]    [65.8289]
E =
    [-4.2170]    [-10.5601]    [6.0994]    [4.1711]
```

可见，训练结果基本满意，要想得到更高的匹配精度，还需要继续对网络进行训练。

此外，还可以采用 `newlind` 函数对上述自适应滤波网直接进行设计，即

```
net=newlind(P,T,Pi);
Y=sim(net,P)
Y =
    [10.0000]    [30.0000]    [50.0000]    [70.0000]
```

可见，利用 `newlind` 函数直接设计的自适应滤波网络实现了零误差输出。

23.4 线性神经网络的局限性

线性神经网络只能反映输入和输出样本向量间的线性映射关系，和感知器神经网络一样，它也只能解决线性可分问题。由于线性神经网络的误差曲面是一个多维抛物面，所以在学习速率足够小的情况下，对于基于最小二乘梯度下降原理进行训练的线性神经网络总可以找到一个最优解。但是，尽管如此，对线性神经网络的训练并不一定总能达到零误差。线性神经网络的训练性能要受网络规则和训练样本集大小的限制。若线性神经网络的自由度（即神经网络所有权值和阈值的个数总和）小于训练样本集中“输入-目标”向量的对数，且各样本向量线性无关，则网络训练不可能达到零误差，而只能得到一个使网络误差最小的解；反之，若网络自由度大于样本集的个数，则会得到无穷多个使网络训练误差为零的解。此外，值得注意的是，线性神经网络的训练和性能要受到学习速率参数的影响，过大的学习速率可能会



导致网络性能发散。

23.5 线性神经网络的 MATLAB 应用举例

【例 23-1】 用直接设计法设计一个简单的线性神经元，使其能够拟合如下所给的输入样本向量和目标向量，其中输入向量为：

```
P=[0.08 -2]
```

目标向量为：

```
T=[0.5 1]
```

本例的 MATLAB 代码如下。

```
clear all
echo on;
pause %敲任意键开始
P=[0.8 -2];
T=[0.5 1];
%设定线性神经元可能的权值和阈值范围
W_range=-1:0.2:1;
B_range=-1:0.2:1;
%根据输入和目标向量绘制线性神经元的误差曲面
ES=errsurf(P,T,W_range,B_range,'purelin');
plotes(W_range,B_range,ES);
%直接设计法设计线性神经网络
net=newlind(P,T);
%对线性神经网络进行仿真
A=sim(net,P)
%计算仿真误差
E=T-A
SE=sse(E)
%根据网络权值和阈值在误差曲面上绘制当前位置点
plotes(W_range,B_range,ES);
plotep(net.iw{1,1},net.b{1},SE);
echo off;
```

窗口显示结果如下：

```
A =
    0.5000    1.0000
E =
     0     0
SE =
     0
```




在利用 `newlin` 函数设计线性神经元之前,可以根据可能的网络权值和阈值范围,并利用函数 `errsurf` 绘制出神经元的误差曲面。本例所绘制的误差曲面如图 23-7 所示,其中越亮的部分误差越小。图 23-7 中所示的亮点为网络设计完成之后根据当前网络权值和阈值并利用 `plotep` 函数绘制的误差位置点,可见误差点位于误差曲面的最低点。

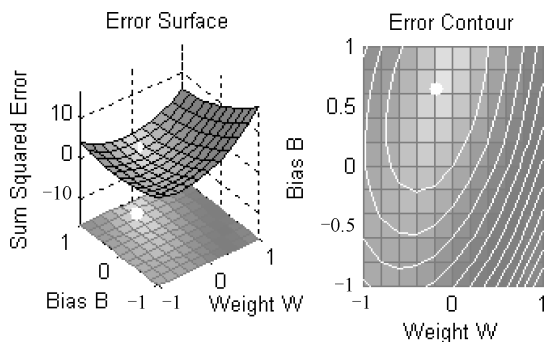


图 23-7 线性神经元的误差曲面和误差点

【例 23-2】 设计并训练一个线性网络,实现从输入向量 P 到输出向量 T 的转换。

```
P=[1 2 4;2 4 8];
T=[0.5 1 -1];
```

从两个向量的结构来看,所设计的线性网络是具有阈值的,并且输入层的神经元为 2 个,输出层的神经元为 1 个。如果仔细观察就会发现输入向量 P 是线性相关的。在这种情况下,线性网络能够解决这个问题,其 MATLAB 代码如下:

```
P=[1 2 4;2 4 8];
T=[0.5 1 -1];
net=newlin(minmax(P),1,0,0.01);
Y=sim(net,P);
net=init(net);
net.trainParam.epochs=2000;
net.trainParam.goal=0.0001;
net=train(net,P,T);
```

网络训练结果为:

```
TRAINB, Epoch 0/2000, MSE 0.75/0.0001.
TRAINB, Epoch 25/2000, MSE 0.576995/0.0001.
.....
TRAINB, Epoch 1975/2000, MSE 0.214286/0.0001.
TRAINB, Epoch 2000/2000, MSE 0.214286/0.0001.
TRAINB, Maximum epoch reached.
```

可见,网络训练第 1000 步至 1500 步时,网络误差发生了轻微的改变,从 1500 步后,网络误差就不再改变,如图 23-8 所示,此时的网络训练误差为 0.214 286 左右。

对网络进行仿真,得到此时的网络输出 Y 。



```
Y=sim(net,P)
```

```
Y =
```

```
0.9286    0.3571   -0.7857
```

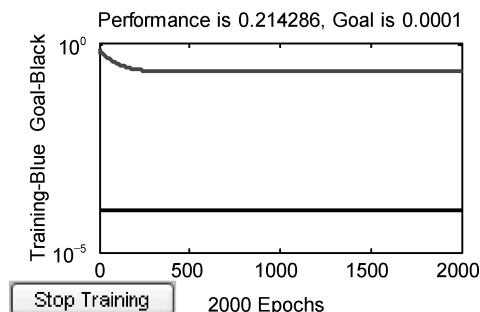


图 23-8 训练误差曲线

此时 Y 与目标向量 T 相比较，发现误差还是比较大的。说明在这种情况下，网络是无法给出完美解决方案的。

【例 23-3】 利用线性神经网络求取非线性问题的最佳线性拟合解。

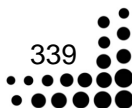
从前面已经知道，线性神经网络只能描述输入、输出间的线性映射关系，当输入、输出间是非线性关系时，利用线性神经网络只能得到输入、输出间的最佳线性拟合解，而不能实现完全的线性拟合。本例我们将采用线性单神经元对如下定义的输入和目标向量进行拟合，即输入向量为 $P=[1 \ 1.2 \ 3 \ -1.2]$ ；目标向量为 $T=[0.4 \ 1 \ 3 \ -0.5]$ 。

由输入向量和目标向量可见，两者之间是非线性关系，即找不到一组权值 W 和阈值 B，使得对所有输入和目标样本向量均满足线性表达式 $T=WP+B$ 。本例中，我们将采用 newlind 函数直接设计法和 train 函数设计法两种方案对线性神经网络进行设计。完整的 MATLAB 代码如下。

```
clear
clf reset
echo on
%定义输入目标和目标向量
P=[1 1.2 3 -1.2];
T=[0.4 1 3 -0.5];
pause
%设定线性神经元可能的权值和阈值范围
W_range=-2:0.2:2;
B_range=-2:0.2:2;
%根据输入和目标向量绘制线性神经元的误差曲面
ES=errsurf(P,T,W_range,B_range,'purelin');
plotes(W_range,B_range,ES);
pause
echo off
disp('1.采用 NEWLIND 函数直接设计线性神经网络');
```



```
disp('2.采用 TRAIN 函数训练并设计线性神经网络');
choice=input('请选择设计方案 (1, 2): ');
if (choice==1)
    echo on
    %用 NEWLIND 函数直接设计线性神经网络
    net=newlind(P,T);
    pause
    %对性线神经网络进行仿真
    A=sim(net,P)
    %计算仿真误差
    E=T-A
    SE=sse(E)
    pause
    %在误差曲面上绘制当前误差位置点
    plotep(net.iw{1,1},net.b{1},SE);
    pause
    echo off
    echo on
    %绘制拟合曲线
    plot(P,T,'r+');
    hold on;
    plot(P,A);
    pause
else
    echo on
    %用 TRAIN 函数训练并设计线性神经网络
    %计算最快的稳定学习速率值
    maxlr=maxlinlr(P,'bias')
    %创建线性神经网络
    net=newlin(minmax(P),1,[0],0.5*maxlr);
    %在误差曲面上选择权值和阈值对线性神经元进行初始化
    plotes(W_range,B_range,ES);
    subplot(1,2,2);
    [net.iw{1,1},net.b{1}]=ginput(1);
    echo off
    SE=sse(T-sim(net,P));
    plotep(ent.iw{1,1},net.b{1},SE);
    echo on
    %对线性神经网络进行训练
    net.trainParam.goal=0.01;
    [net,tr]=train(net,P,T);
    pause;
```





```
%对线性神经网络进行仿真
A=sim(net,P)
E=T-A
SE=sse(E)
pause
%在误差曲面上绘制当前误差位置点
plotep(net.iw{1,1},net.b{1},SE);
pause
%绘制误差变化曲线
plotperf(tr,net.trainParam.goal);
pause
%绘制拟合曲线
plot(P,T,'b*');
hold on;
plot(P,A);
pause
end
echo off
```

按任意键执行代码，命令窗口显示如下内容给用户选择设计方式。

(1) 采用 NEWLIND 函数直接设计线性神经网络

(2) 采用 TRAIN 函数训练并设计线性神经网络

请选择设计方案 (1, 2): 1

在此选择方案 1 后按回车键显示结果为:

```
A =
    0.9750    1.1394    2.6191   -0.8336
E =
   -0.5750   -0.1394    0.3809    0.3336
SE =
    0.6064
```

实践证明，两种方案均未能达到拟合零误差的效果，但两种方案都得到了相同的最佳拟合结果，图 23-9 所示为拟合曲线。此外，图 23-10 给出了所设计的线性神经元的误差曲面和误差点位置。

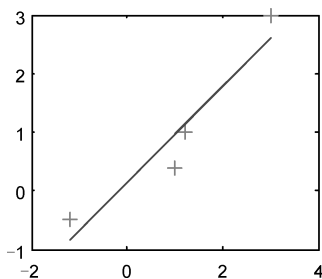


图 23-9 线性神经元最佳拟合曲线

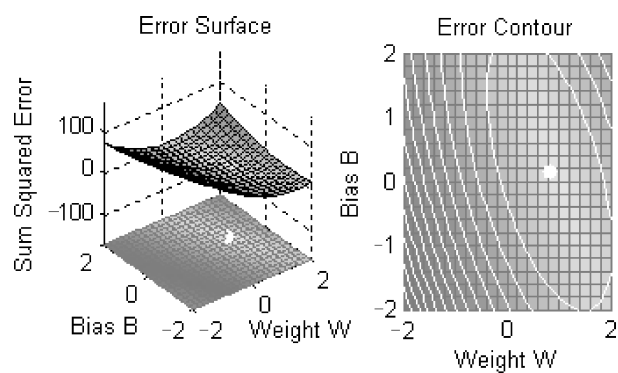


图 23-10 线性神经元的误差曲面和误差点

第 24 章 神经网络工具箱函数及实例分析



24.1 传递函数及其导函数

神经网络的传递函数及其导函数如表 24-1 所示。

表 24-1 传递函数及其导数

函数名称	功 能	函数名称	功 能
compet	竞争传递函数	tribas	三角基传递函数
hardlim	硬限幅传递函数	dhardlim	硬限幅传递函数的导函数
hardlims	对称硬限幅传递函数	dhardlims	对称硬限幅传递函数的导函数
logsig	对数 Sigmoid 传递函数	dlogsig	对数 Sigmoid 传递函数的导函数
poslin	正线性传递函数	dposlin	正线性传递函数的导函数
purelin	纯线性传递函数	dpurelin	纯线性传递函数的导函数
radbas	离斯径向基传递函数	dradbas	离斯径向基传递函数的导函数
satlin	饱和线性传递函数	dsatlin	饱和线性传递函数的导函数
satlins	对称饱和线性传递函数	dsatlins	对称饱和线性传递函数的导函数
softmax	softmax 传递函数	dtansig	正切 Sigmoid 传递函数的导函数
tansig	正切 Sigmoid 传递函数	dtribas	三角基传递函数的导函数

24.1.1 传递函数

1) compet

应用 竞争传递函数。

格式 `A=compet(N)`

`info=compet(code)`

解析 竞争传递函数 `compet` 用来寻找输入向量 `N` 中的最大元素，并把相应神经元的输出设为 1，其余输出设为 0。输出为 1 的神经元称为获胜神经元。

函数调用形式 `info=compet(code)` 返回有关的函数信息，`code` 字符串取值如下所述。

'deriv': 返回导函数的名称，函数 `compet` 没有导函数；



'name': 返回传递函数的全称;

'output': 返回传递函数的输出范围;

'active': 返回传递函数的活跃输出范围。

newc 和 newpnn 函数建立的网络都采用 compet 函数作为传递函数。

【例 24-1】 下面给出一组 MATLAB 代码, 可以演示该函数的功能和运行机理。

```
N=[0;1;-0.5;0.5];
A=compet(N)
info=compet('name')
subplot(2,1,1)
bar(N),ylabel('N')
subplot(2,1,2)
bar(A),ylabel('A')
```

向量 A 和 N 如图 24-1 所示。

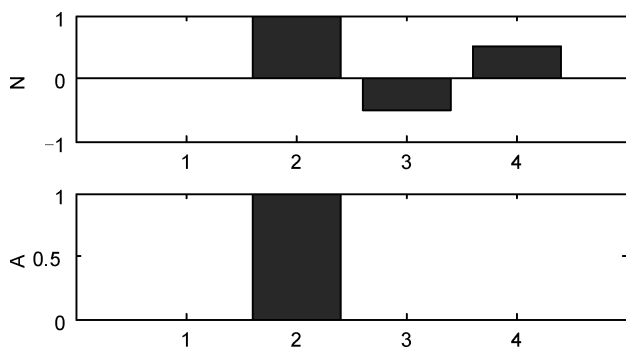


图 24-1 函数 compet 的向量 A 和 N

运行结果为:

```
A =
(2,1)      1
info =
Competitive
```

2) softmax

功能 softmax 传递函数。

格式 $A = \text{softmax}(N)$

info=softmax (code)

解析 softmax 传递函数的计算公式用 MATLAB 语句表示为:

$$a = \exp(n) / \sum(\exp(n))$$

其参数含义参见 compet 函数, 与 compet 不同的是, 参数 A 为函数返回向量, 各元素在区间[0, 1]之间, 且向量结构与 N 一致。该函数也没有导函数。



【例 24-2】 利用如下一组代码演示该函数的功能和运行机理。

```
N=[0;1;-0.5;0.5];  
A=softmax(N)  
info=softmax('name')  
subplot(2,1,1)  
bar(N),ylabel('N')  
subplot(2,1,2)  
bar(A),ylabel('A')
```

运行结果为：

```
A =  
    0.1674  
    0.4551  
    0.1015  
    0.2760  
  
info =  
Soft Max
```

向量 A 和 N 如图 24-2 所示。

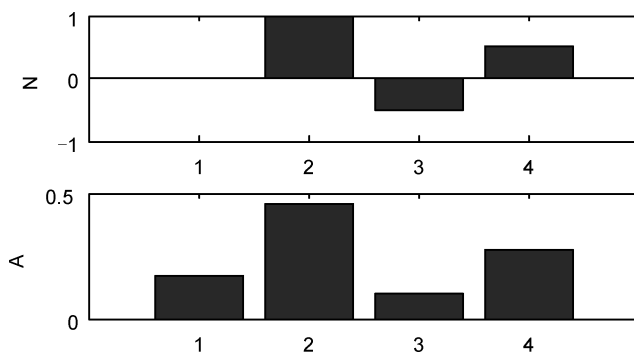


图 24-2 函数 softmax 的向量 A 和 N

3) hardlim

应用 硬限幅传递函数。

格式 $A = \text{hardlim}(N)$

$\text{info} = \text{hardlim}(\text{code})$

解析 硬限幅传递函数 hardlim 把函数输入向量 N 中各元素的取值强迫限制为 1 或 0, 当输入大于或等于 0 时, 神经元的输出为 1, 否则输出为 -1。

newp 函数建立的网络可以采用 hardlims 函数作为传递函数。

该函数的原型函数为：

$$\text{hardlim}(n) = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$



4) hardlims

应用 对称硬限幅传递函数。

格式 $A = \text{hardlims}(N)$

$\text{info} = \text{hardlims}(\text{code})$

解析 对称硬限幅传递函数 hardlims 把函数输入向量 N 中各元素的取值强迫限制为 1 或 -1, 当输入大于或等于 0 时, 神经元的输出为 1, 否则输出为 0。

newp 函数建立的网络可以采用 hardlim 函数作为传递函数。

该函数的原型函数为:

$$\text{hardlims}(n) = \begin{cases} 1 & n \geq 0 \\ -1 & n < 0 \end{cases}$$

由此可见, 以上两个函数可以实现神经网络的分类和判断功能。

【例 24-3】 利用 MATLAB 给出一个数组, 调用函数 hardlim 与 hardlims 对其进行分类。MATLAB 代码如下。

```
%以 0.1 为步长, 建立一个数组
N=-6:0.1:6;
A1=hardlim(N);
A2=hardlims(N);
plot(N,A1,'ro');
hold on
plot(N,A2,'b+');
hold on
```

运行结果如图 24-3 所示。

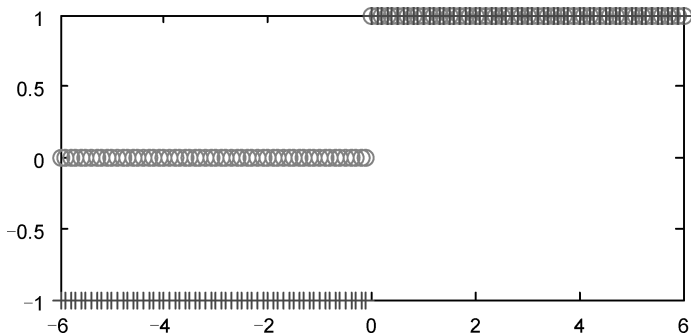


图 24-3 利用传递函数进行分类的结果

图中加号部分是函数 hardlims 的分类结果; 圆圈部分是 hardlim 的分类结果。可以看出两个函数都成功地对数组进行了分类。

5) logsig

功能 对数 Sigmoid 传递函数。

格式 $A = \text{logsig}(N)$

$\text{info} = \text{logsig}(\text{code})$



解析 对数 Sigmoid 传递函数的计算公式用 MATLAB 语句表示为:

```
logsig(n)=1/(1+exp(-n))
```

newff 和 newcf 函数建立的网络都可以采用该函数作为传递函数。

【例 24-4】 下面举例以一组简单的代码产生一个对数 S 型的传递函数。

```
N=-10:0.1:10;  
A=logsig(N)  
plot(N,A)  
grid
```

运行结果如图 24-4 所示。可见, 函数 `logsig` 可将神经元的输入 (范围为整个实数集) 映射到区间 (0, 1) 中, 又由于该函数为可微函数, 因此非常适合于利用 BP 算法训练神经网络。

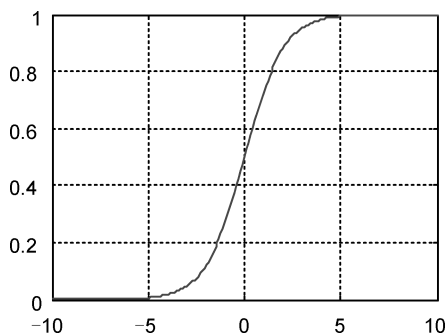


图 24-4 logsig 函数运行结果

6) poslin

功能 正线性传递函数。

格式 `A=poslin(N)`

`info=poslin(code)`

解析 对于输入向量 `N` 中的正元素或 0, 正线性传递函数按照原始数值输出; 对于负元素, 函数将输出 0。

【例 24-5】 求正线性传递函数对下列输入向量的输出。

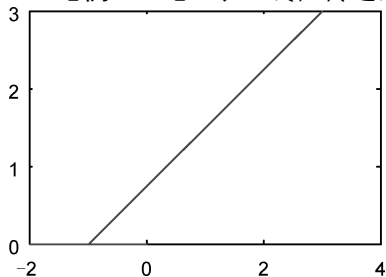


图 24-5 poslin 的运行结果

```
N=[-2;0;-1;3];  
A=poslin(N);  
A'  
plot(N,A)
```

运行结果如下及如图 24-5 所示。

```
ans =  
0 0 0 3
```

7) purelin

应用 纯线性传递函数。



格式 `A=purelin(N)`

`info=purelin(code)`

解析 纯线性传递函数把输入 N 按照原始数据输出。`newlin` 和 `newlind` 函数建立的网络都可以采用该函数作为传递函数。

【例 24-6】 下面以一组简单的代码产生一个线性 S 型传递函数。

```
N=-10:0.1:10;
A=purelin(N);
plot(N,A)
grid
```

运行结果如图 24-6 所示。

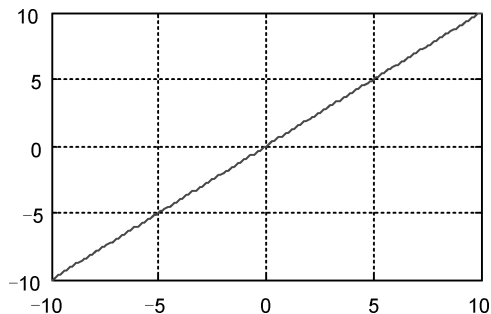


图 24-6 purelin 函数运行结果

8) radbas

应用 高斯径向基传递函数。

格式 `A=radbas(N)`

`info=radbas(code)`

解析 高斯径向基传递函数的计算公式用 MATLAB 语句表示为：

$$a = \exp(-n.^2)$$

`newpnn` 和 `newgrann` 函数建立的网络都可以采用该函数作为传递函数。

【例 24-7】 下面以一组简单的代码产生一个高斯径向基传递函数。

```
N=-10:0.1:10;
A=radbas(N);
plot(N,A)
grid
```

运行结果如图 24-7 所示。

9) satlin

应用 饱和线性传递函数。

格式 `A=satlin(N)`

`info=satlin(code)`

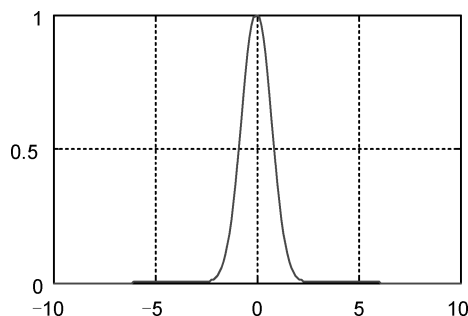


图 24-7 radbas 函数运行结果

解析 饱和线性传递函数把输入向量 N 中位于 $[0, 1]$ 区间内的元素按照原始数据输出；对于小于 0 的元素，则输出 0；对于大于 1 的元素，则输出 1。

【例 24-8】 求饱和线性传递函数对下列输入向量的输出。

```
N=[-3;-1;0.5;1;3];
A=satlin(N);
A'
ans =
      0      0  0.5000  1.0000  1.0000
```

10) satlins

应用 对称饱和线性传递函数。

格式 $A = \text{satlins}(N)$

$\text{info} = \text{satlins}(\text{code})$

解析 对称饱和线性传递函数把输入向量 N 中位于 $[-1, 1]$ 区间内的元素按照原始数据输出；对于小于 -1 的元素，则输出 -1；对于大于 1 的元素，则输出 1。

newhop 函数建立的网络采用该函数作为传递函数。

【例 24-9】 求对称饱和线性传递函数对下列输入向量的输出。

```
N=[-3;-1;0.5;1;3];
A=satlins(N);
A'
ans =
 -1.0000 -1.0000  0.5000  1.0000  1.0000
```

11) tansig

应用 正切 Sigmoid 传递函数。

格式 $A = \text{tansig}(N)$

$\text{info} = \text{tansig}(\text{code})$

解析 **tansig** 传递函数的计算公式用 MATLAB 语句表示为：

$$n = 2 / (1 + \exp(-2 * n)) - 1$$

newff 和 **newcf** 函数建立的网络都可以采用该函数作为传递函数。



【例 24-10】 以下代码可绘制一个双曲正切 S 型传递函数。

```
N=-10:0.1:10;
A=tansig(N);
plot(N,A)
grid
```

运行结果如图 24-8 所示。

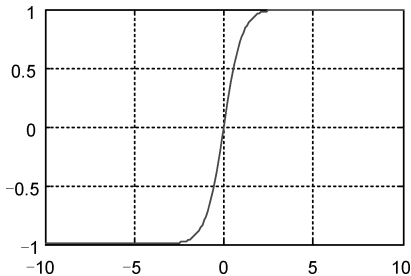


图 24-8 tansig 函数运行结果

12) tribas

应用 三角基传递函数。

格式 $A = \text{tribas}(N)$

$\text{info} = \text{tribas}(\text{code})$

解析 当输入向量 N 中的元素位于 $[-1, 1]$ 区间内时，tribas 传递函数的计算公式用 MATLAB 语句表示为：

$$n = 1 - \text{abs}(n)$$

当元素在该区间外时，函数输出为 0。

【例 24-11】 求 tribas 传递函数对下列输入向量的输出。

```
N=[-3;-1;-0.8;0.5;0.8;1;3];
A=tribas(N);
A'
ans =
```

```
0      0      0.2000      0.5000      0.2000      0      0
```

24.1.2 传递函数的导函数

1) dhardlim

功能 硬限幅传递函数的导函数。

格式 $dA_dN = \text{dhardlim}(N, A)$

解析 该函数用来计算硬限幅函数的输出 A 相对于输入 N 的导数。

【例 24-12】 利用 MATLAB 给出一个数组，调用函数 dhardlim 对其进行分类，MATLAB



代码如下：

```
%以 0.1 为步长，建立一个数组
N=-6:0.1:6;
A1=hardlim(N);
dA_dN=dhardlim(N,A1)
plot(N,A,'r+');
hold on
```

运行效果如图 24-9 所示。

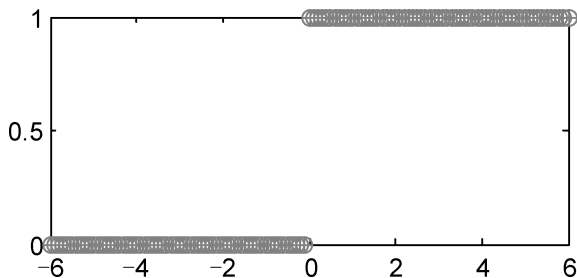


图 24-9 dhardlim 函数运行结果

2) dhardlims

功能 对称硬限幅度传递函数的导函数。

格式: $dA_dN = dhardlims(N, A)$

解析 该函数用于计算对称硬限幅函数的输出 A 相对于输入 N 的导数。

【例 24-13】 求对称硬限幅函数的输出 A 相对于输入 N 的导数。

```
N=[-3;-1;-0.8;0.5;0.8;1;3];
A=hardlims(N);
dA_dN=dhardlim(N,A);
dA_dN'
ans =
    0    0    0    0    0    0    0
```

3) dlogsig

功能 对数 Sigmoid 传递函数的导函数。

格式 $dA_dN = dlogsig(N, A)$

解析 该函数用于计算对数 Sigmoid 函数的输出 A 相对于输入 N 的导数。

【例 24-14】 现有某 BP 网络层的 3 个神经元，其传递函数均为 S 型的对数函数，试利用函数 `logsig` 求解输出对输入的导数。

```
N=[-3;-1;-0.8;0.5;0.8;1;3];
N=[0.2;0.9;-0.6;-2];
A=logsig(N)
dA_dN=logsig(N,A)
```



运行结果如下。

```
A =  
    0.5498  
    0.7109  
    0.3543  
    0.1192  
dA_dN =  
    0.6791  
    0.8335  
    0.4389  
    0.1323
```

4) dposlin

应用 正线性传递函数的导函数。

格式 `dA_dN=dposlin(N,A)`

解析 该函数用于计算正线性函数的输出 A 相对于输入 N 的导数。

【例 24-15】 求正线性传递函数的输出对下列输入向量的导数。

```
N=[-1;0;0.5];  
A=poslin(N);  
dA_dN=dposlin(N,A)  
dA_dN =  
     0  
     1  
     1
```

5) dpurelin

应用 纯线性传递函数的导函数。

格式 `dA_dN=dpurelin(N,A)`

解析 该函数用于计算纯线性函数的输出 A 相对于输入 N 的导数。

【例 24-16】 求纯线性传递函数的输出对下列输入向量的导数。

```
N=[-1;0;0.5];  
A=purelin(N);  
dA_dN=dpurelin(N,A)  
dA_dN =  
     1  
     1  
     1
```

6) dradbas

功能 高斯径向基传递函数的导函数。



格式 $dA_dN = \text{dradbas}(N, A)$

解析 该函数用于计算高斯径向基函数的输出 A 相对于输入 N 的导数。

【例 24-17】 下面以一组简单的代码产生一个高斯径向基函数传递函数的导函数。

```
N=-10:0.1:10;
A=radbas(N);
dA_dN=dradbas(N,A);
plot(N,dA_dN)
grid
```

运行结果如图 24-10 所示。

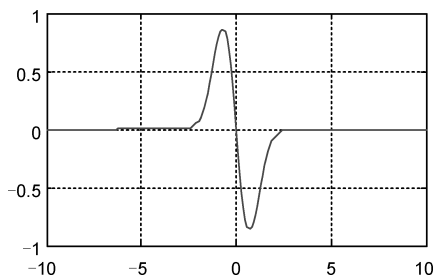


图 24-10 dradbas 函数运行结果

7) dsatlin

功能 饱和线性传递函数的导函数。

格式 $dA_dN = \text{dsatlin}(N, A)$

解析 该函数用于计算饱和线性函数的输出 A 相对于输入 N 的导数。

【例 24-18】 求饱和线性传递函数的输出对下列输入向量的导数。

```
N=[-1;0;0.5;2];
A=satlin(N);
dA_dN=dsatlin(N,A)
dA_dN =
    0
    1
    1
    0
```

8) dsatlins

功能 对称饱和线性传递函数的导函数。

格式 $dA_dN = \text{dsatlins}(N, A)$

解析 该函数用于计算对称饱和线性函数的输出 A 相对于输入 N 的导数。

【例 24-19】 成本求对称饱和线性传递函数的输出对下列输入向量的导数。

```
N=[-3;-1;0;0.5;2];
A=satlins(N);
```




```
dA_dN=dsatlin(N,A)
dA_dN =
    0
    0
    1
    1
    0
```

9) dtansig

功能 tansig 传递函数的导函数。

格式 dA_dN=dtansig(N,A)

解析 该函数用于计算 tansig 函数的输出 A 相对于输入 N 的导数。

【例 24-20】 以下代码可绘制一个双曲正切 S 型传递函数的导函数。

```
N=-10:0.1:10;
A=tansig(N);
dA_dN=dradbas(N,A);
plot(N,dA_dN)
grid
```

运行结果如图 24-11 所示。

10) dtribas

功能 三角基传递函数的导函数。

格式 dA_dN=dtribas(N,A)

解析 该函数用于计算三角基函数的输出 A 相对于输入 N 的导数。

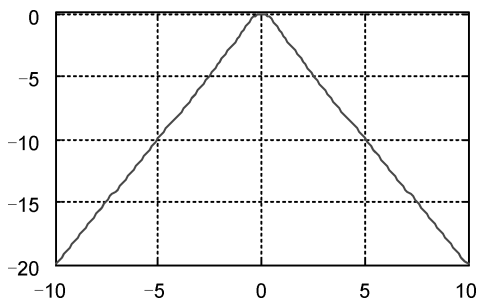


图 24-11 dtansig 的运行结果

【例 24-21】 求 triabas 传递函数的输出对下列输入向量的导数。

```
N=[-2;0.5;0.8;2];
A=tribas(N);
dA_dN=dtribas(N,A)
dA_dN =
    0
   -1
```

-1
0

24.2 距离函数

自组织映射网络的距离函数如表 24-2 所示。

表 24-2 距离函数

函数名称	功 能	函数名称	功 能
boxdist	Box 距离函数	dist	欧氏距离加权函数
linkdist	连接距离函数	mandist	曼哈顿距离加权函数

1) boxdist

应用 该函数为 Box 距离函数。

格式 `d=boxdist(pos)`

解析 在给定神经网络某层神经元的位置后，可利用该函数计算神经元之间的距离。该函数通常用于结构函数 `gridtop` 的神经网络层，其参数含义如下所述。

pos: 神经元位置的 $N \times S$ 维矩阵；

d: 函数返回值，神经元距离的 $S \times S$ 维矩阵。

函数的运算原理为 $d(i, j) = \max \|P_i - P_j\|$ 。其中， $d(i, j)$ 表示距离矩阵中的元素； P_i 表示位置矩阵的第 i 列向量。

【例 24-22】 以下 MATLAB 代码可以演示该函数的运算规则。

```
%产生一个 3×3 的矩阵
pos=rand(3,3)
d=boxdist(pos)
```

运行结果为：

```
pos =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

d =
         0    0.6602    0.4937
    0.6602         0    0.8728
    0.4937    0.8728         0
```

2) linkdist

应用 该函数为连接距离函数。



格式 `d=linkdist(pos)`

解析 在给定神经元的位置后，该函数可用于计算神经元之间的距离，其参数含义见 `boxdist` 函数。

函数的运算原理为：

$$d(i, j) = \begin{cases} 0 & \text{如果 } i = j \\ 1 & \text{如果 } \text{sum}((P_i - P_j)^2)^{\frac{1}{2}} \leq 1 \\ 2 & \text{如果存在 } k, \text{使得 } d(i, k) = d(k, j) = 1 \\ 3 & \text{如果存在 } k_1, k_2, \text{使得 } d(i, k_1) = d(k_1, k_2) = d(k_2, j) = 1 \\ N & \text{如果存在 } k_1, k_2, \dots, k_N, \text{使得 } d(i, k_1) = d(k_1, k_2) = \dots = d(k_N, j) = 1 \\ S & \text{其他} \end{cases}$$

【例 24-23】下面用一组代码演示该函数的运算原理。

```
%产生一个 3×3 的矩阵
pos=rand(3,3);
d=linkdist(pos)
```

运行结果为：

```
d =
    0     1     1
    1     0     1
    1     1     0
```

3) dist

应用 欧氏距离加权函数。

格式 `Z=dist(W,P)`

`df=dist('deriv')`

`d=dist(pos)`

解析 通过对输入进行加权得到加权后的输入，其参数含义如下。

W: $S \times R$ 维的权值矩阵；

P: Q 组输入（列）向量的 $R \times Q$ 维矩阵；

Z: $S \times Q$ 维的距离矩阵；

`df=dist('deriv')`：返回值为空，因为该函数不存在导函数；

其他参数含义请参见 `boxdist` 函数。

函数的运算规则为 $d = \sqrt{\text{sum}(x - y)^2}$ ，其中 x 和 y 分别为列向量。

【例 24-24】下面用一组代码演示该函数的运行机理。

```
W=rand(3,3);
P=rand(3,1);
pos=rand(3,3);
Z=dist(W,P)
d=dist(pos)
```



运行结果为：

```
Z =  
    0.7306  
    0.8739  
    0.7668  
d =  
     0    0.2340    0.7917  
    0.2340     0    0.6875  
    0.7917    0.6875     0
```

4) mandist

应用 为 Manhattan 距离权函数。

格式 $Z = \text{mandist}(W, P)$

$df = \text{mandist}('deriv')$

$d = \text{mandist}(\text{pos})$

各参数含义参见 `dist`。

函数的运算原理为 $d = \text{sum}[\text{abs}(X - Y)]$ ，其中 X 和 Y 为两个向量。

【例 24-25】 利用下面一组代码进行演示 `mandist` 函数。

```
pos=rand(3,3);  
d=mandist(pos)
```

运行结果为：

```
d =  
     0    0.5246    0.4848  
    0.5246     0    0.2317  
    0.4848    0.2317     0
```

24.3 权值函数及其导函数

加权函数如表 24-3 所示，这类函数确定了神经网络每层的输入向量和神经元权值矩阵通过何种计算方式得到神经元的传递函数的加权输入量。

表 24-3 加权函数及其导函数

函数名称	功 能	函数名称	功 能
<code>dotprod</code>	内积加权函数	<code>normprod</code>	规范化内积加权函数
<code>negdist</code>	负欧氏距离加权函数	<code>ddotprod</code>	内积加权函数的导函数



24.3.1 权值函数

1) dotprod

应用 内积加权函数。

格式 $Z = \text{dotprod}(W, P)$

$df = \text{dotprod}('deriv')$

解析 它求得权值与输入之间的点积作为加权输入，其参数含义如下。

W : $S \times R$ 维的权值矩阵；

P : Q 组 R 维的输入向量；

Z : Q 组 R 维的 W 与 P 的点积；

$df = \text{dotprod}('deriv')$: 返回函数的导数。

【例 24-26】 建立两个矩阵（或向量），演示函数 `dotprod` 的功能。

```
% 建立一个 4×3 的矩阵，所有元素都小于 1
W=rand(4,3)
% 建立一个 3×1 的矩阵，所有元素都小于 1
P=rand(3,1);
Z=dotprod(W,P);
P'
Z'
```

输出结果为：

```
W =
    0.2259    0.6405    0.6808
    0.5798    0.2091    0.4611
    0.7604    0.3798    0.5678
    0.5298    0.7833    0.7942
ans =
    0.0592    0.6029    0.0503
ans =
    0.4338    0.1835    0.3025    0.5435
```

2) negdist

应用 负欧氏距离加权函数。

格式 $Z = \text{negdist}(W, P)$

$df = \text{negdist}('deriv')$

解析 `negdist` 函数用于计算两向量间的负欧氏距离，其运算原理为 $d = -\sum((x - y)^2)^{\frac{1}{2}}$ ，

其中， d 为向量 x 与 y 之间的负欧氏距离。



函数调用时各参数见 dotprod 函数，negdist 没有导函数。

【例 24-27】 设定一个权值矩阵 W 和输入向量 P ，计算其负欧氏距离加权输入。

```
W=[0.9 0.5 0.7;0.1 0.6 0.7];
P=[0.3 0 1]';
Z=negdist(W,P)
```

输出为：

```
Z =
    -0.8367
    -0.7000
```

3) normprod

应用 规则化内积加权函数。

格式 $Z = \text{normprod}(W, P)$

$df = \text{normprod}('deriv')$

解析 normprod 函数用于计算两向量之间的规则化内积，其计算原理为 $Z = W \times P / \text{sum}(P)$ 。其中， W 为权值向量， P 为输入向量， Z 为规则化内积。

函数的参数含义参见 dotprod 函数。normprod 函数没有导函数。

【例 24-28】 设定权值矩阵 W 和输入向量 P ，计算其规则化内积的加权输入。

```
W=[0.9 0.5 0.7;0.1 0.6 0.7];
P=[0.3 0 0.8]';
Z=normprod(W,P)
```

输出结果为：

```
Z =
    0.7545
    0.5364
```

24.3.2 权值函数的导函数

ddotprod

应用 内积加权函数的导函数。

格式 $dZ_dP = \text{ddotprod}('p', W, P, Z)$

$dZ_dW = \text{ddotprod}('w', W, P, Z)$

解析 ddotprod 函数用于计算内积函数的输出对输入向量的导数。

在函数的调用形式 $dZ_dP = \text{ddotprod}('p', W, P, Z)$ 中， W 为权值矩阵， P 为该层网络的输入向量， Z 为内积矩阵，函数返回内积函数输出对输入向量的导数。函数的调用形式 $dZ_dW = \text{ddotprod}('w', W, P, Z)$ 返回内积函数的输出量对权值矩阵的导数。

【例 24-29】 设定权值矩阵 W 和输入向量 P ，计算内积加权输入及其导数。



```
W=[0.9 0.5 0.7;0.1 0.6 0.7];
P=[0.3 0 0.8]';
Z=dotprod(W,P)
dZ_dP=ddotprod('p',W,P,Z)
dZ_dW=ddotprod('w',W,P,Z)
```

输入结果为：

```
Z =
    0.8300
    0.5900
dZ_dP =
    0.9000    0.5000    0.7000
    0.1000    0.6000    0.7000
dZ_dW =
    0.3000
         0
    0.8000
```

24.4 结构函数

结构函数如表 24-4 所示。

1) hextop

应用 该函数为六角层结构函数。

格式 pos=hextop(dim1, dim2, ..., dimN)

解析 dimi 维数为 i 的层的长度；pos 由 N 个并列向量组成的 $N \times S$ 维矩阵，其中， $S = \text{dim1} \times \text{dim2} \times \cdots \times \text{dimN}$ 。

表 24-4 结构函数

函数名称	功 能
hextop	六角层结构函数
gridtop	网格层结构函数
randtop	随机层结构函数

【例 24-30】 利用该函数创建一个二维的神经网络层，共有 35 个神经元，分布在一个 7×5 的六角晶格上，其代码如下。

```
pos=hextop(7,5);
plotsom(pos)
```

运行结果如图 24-12 所示。

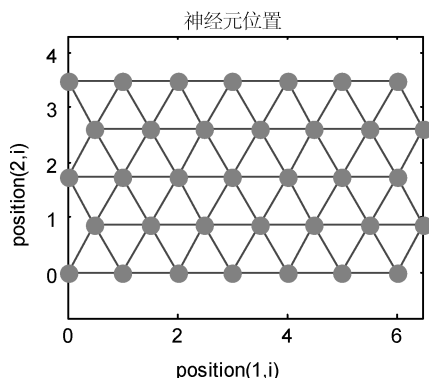


图 24-12 函数 hextop 产生的 35 个神经元的分布位置

2) gridtop

应用 为网格层结构函数。

格式 `pos=gridtop(dim1, dim2, ..., dimN)`

解析 各参数含义见 hextop 函数。

【例 24-31】 利用该函数创建一个二维的神经网络层，共有 35 个神经元，分布在一个 7×5 的网格上，代码如下。

```
pos=gridtop(7,5);
plotsom(pos)
```

运行结果如图 24-13 所示。

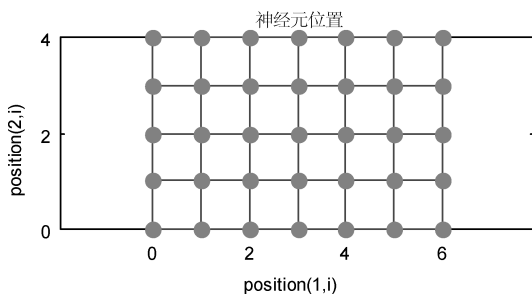


图 24-13 函数 gridtop 产生的 35 个神经元的分布位置

3) randtop

应用 为随机层结构函数。

格式 `pos=randtop(dim1, dim2, ..., dimN)`

解析 各参数含义见 hextop 函数。

【例 24-32】 利用该函数创建一个二维的神经网络层，共有 35 个神经元，分布在一个 7×5 的随机格上，代码如下。

```
pos=randtop(7,5);
plotsom(pos)
```




运行结果如图 24-14 所示。

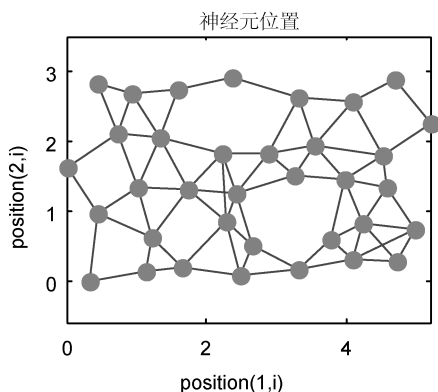


图 24-14 函数 randtop 产生的 35 个神经元的分布位置

24.5 分析函数

分析函数如表 24-5 所示。

1) errsurf

应用 计算单输入神经元的误差曲面。

格式 $E = \text{errsurf}(P, T, WV, BV, F)$

解析 该函数可用于计算单输入神经元在给定的权值范围向量和阈值范围向量情况下的网络误差。函数的输入为神经元向量 P 、目标向量 T 、权值范围向量 WV 、阈值范围向量 BV 以及神经元传递函数 F ，函数返回误差曲面各点的误差 E 。

表 24-5 分析函数

函数名称	功 能
errsurf	计算单输入神经元的误差曲面
maxlinlr	取线性神经网络的最大学习速率

【例 24-33】 下面的一组代码可以分析一个 BP 网络中某个神经元的误差，并绘制出其误差曲面与轮廓线。

```
P=[-6.1 6 -4.1 -4 4 4.1 -6 6.1];
T=[0 1 0.89 0.92 0.02 0.04 0 1];
WV=-1.5:0.1:1.5;
BV=-3:0.2:3;
ES=errsurf(P,T,WV,BV,'logsig');
%绘制误差曲面和等高线，如图 24-15（a）和图 24-15（b）所示
plotes(WV,BV,ES,[60 30])
W=0;
```

```
B=0;
E=sse(T-logsig(W*P+B));
plotep(W,B,E);
```

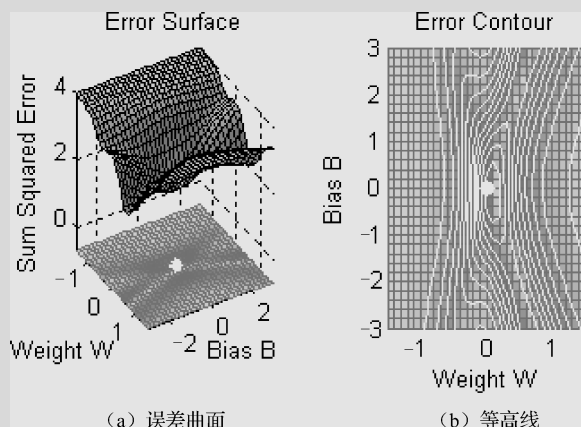


图 24-15 误差曲面和等高线

2) maxlinlr

功能 求取线性神经网络的最大学习速率。

格式 `lr=maxlinlr(P)`

`lr=maxlinlr(P, 'bias')`

解析 `maxlinlr` 函数可以根据线性神经网络的输入向量 **P** 设定网络的学习速率。当网络神经元没有阈值时，使用函数形式 `lr=maxlinlr(P)` 来获得权值的学习速率；当网络神经元有阈值时，使用形式 `lr=maxlinlr(P, 'bias')` 得到权值和阈值的学习速率。

【例 24-34】 下面一组代码可在给定输入 **P** 的情况下，分“带阈值”和“不带阈值”两种情况求得该线性层所需的最大学习速率。

```
P=[1 2 -3 7;0.2 4 10 7];
lr1=maxlinlr(P,'bias')
lr2=maxlinlr(P)
```

运行结果为：

```
lr1 =
    0.0057
lr2 =
    0.0058
```

24.6 转换函数

转换函数如表 24-6 所示。



表 24-6 转换函数

函数名称	功 能
sp2narx	转换串联输入为平行输入
ind2vec	将数据索引转换为向量组
vec2ind	ind2vec 的逆函数
cell2mat	将矩阵构成的单元数组组合成矩阵
mat2cell	将矩阵分裂为由子矩阵构成的单元数组
combvec	建立向量所有组合的矩阵
con2seq	将并行向量转换为串行向量
concur	复制阈值向量并构成矩阵
mimmax	求矩阵每行的范围
normr	正则化矩阵的行
normc	正则化矩阵的列
pnormc	伪正则化矩阵的列
quant	将实数量化为某一数值的整数倍
sumsq	求矩阵的平方和
seq2con	将串行向量转变为并行向量

1) sp2narx

应用 转换串联输入为平行输入

格式 net=sp2narx (net)

解析 其中 net=sp2narx 用于把一个串联输入非线性回归神经网络转换为平行输入；net 原来的串联输入非线性回归神经网络。

2) ind2vec

应用 用于将数据索引转换为向量组。

格式 vec=ind2vec (ind)

解析 ind 为数据索引列向量；vec 为函数返回值，一个稀疏矩阵，每行只有一个 1，矩阵的行数等于数据索引的个数，列数等于数据索引中的最大值。

【例 24-35】 可通过下面的一组代码演示 ind2vec 的计算原理。

```
ind=[3 5 7 9];
vec=ind2vec(ind)
```

上述代码的第一行定义了一个数据索引列向量，第二行将其转换为向量组，运行结果为：

```
vec =
(3,1)    1
(5,2)    1
(7,3)    1
(9,4)    1
```



可以看出,结果应该是一个 4×3 的矩阵,其中 (x,y) 是与数据索引列向量相对应的。 x 为数据索引值, y 表示 x 在数据索引中的位置,由此组成了输出矩阵。矩阵中没有填满的部分需要用 0 补齐。

3) vec2ind

该函数用于将向量组转换为数据索引,与 `ind2vec` 是互逆的。

4) cell2mat

应用 将矩阵构成的单元数组组合成矩阵。

格式 `m=cell2mat(c)`

解析 `cell2mat` 函数可以把矩阵构成的单元数组重组为矩阵。函数的输入 `c` 是一个单元数组,函数返回由 `c` 中各元素构成的矩阵。

【例 24-35】

```
c={ [1] [3] [4;8] [5;7] };  
m=cell2mat(c)
```

输出为:

```
m =  
     1     3  
     4     5  
     8     7
```

5) combvec

应用 建立向量所有组合的矩阵。

格式 `a=combvec(a1,a2,...,aN)`

解析 `combvec` 函数将建立输入矩阵 `a1` 到 `aN` 中各列向量的所有组合,并返回由这些组合构成的矩阵。

【例 24-36】

```
a1=[1 3;5 7];  
a2=[2;4];  
a3=combvec(a1,a2)
```

输出为:

```
a3 =  
     1     3  
     5     7  
     2     2  
     4     4
```

6) con2seq

应用 将并行向量转变为串行向量。



格式 `s=con2seq(b)`

`s=con2seq(b,TS)`

解析 在神经网络工具箱中，矩阵用于存储神经网络的并行输入向量，单元数组用于存储网络的串行输入向量。`con2seq` 函数可以实现网络并行输入向量到串行输入向量的转换。

在函数的调用 `s=con2seq(b)` 中，函数输入矩阵 b ，返回单元数组 s ， s 中的每一个元素为矩阵 b 的列向量。在函数的调用 `s=con2seq(b,TS)` 中，函数输入 b 为 $N \times 1$ 维单元数组， b 中的每一个元素是由 $M \times TS$ 列并行输入向量构成的矩阵， TS 为网络的工具步数；函数返回 $N \times TS$ 维单元数组 s ， s 中的每一元素是由 M 列并行输入向量构成的矩阵。

【例 24-37】

```
P1=[1 3 5];
P2=con2seq(P1)
```

输出为：

```
P2 =
    [1]    [3]    [5]
```

7) concur

应用 复制阈值向量并构成矩阵。

格式 `concur(B,Q)`

解析 `concur` 函数的输入 B 为阈值向量， Q 为复数次，函数返回的矩阵是由 Q 个原阈值向量构成的矩阵。

【例 24-38】

```
B=[2;4;6];
concur(B,3)
```

输出为：

```
ans =
     2     2     2
     4     4     4
     6     6     6
```

8) mat2cell

应用 将矩阵分裂为由子矩阵构成的单元数组。

格式 `cell=mat2cell(M,R,C)`

解析 `mat2cell` 函数对输入矩阵 M 进行分裂，分裂后各子矩阵的行数由向量 R 决定，列数由向量 C 决定。函数返回由各子矩阵构成的单元数组 `cell`。

【例 24-39】

```
M=[1 4 7;2 5 8;3 6 9];
C=mat2cell(M,[2 1],[1 2])
```

输出为：



```
C =  
    [2x1 double]    [2x2 double]  
    [          3]    [1x2 double]
```

9) minmax

应用 求矩阵每行的范围。

格式 PR=minmax(P)

解析 minmax 函数求取输入矩阵 P 中每一行向量的取值范围，并返回各行中最小值和最大值构成的范围矩阵 PR。

【例 24-40】

```
P=[-1 -5 6 1;7 2 -1 4];  
PR=minmax(P)
```

输出为：

```
PR =  
    -5     6  
    -1     7
```

10) normr

应用 正则化矩阵的行。

格式 N=normr(M)

解析 normr 函数对其输入矩阵 M 进行正则化，函数返回的正则化矩阵 N 与原始矩阵维数相同，矩阵每一行元素的平方和变为 1，但各元素之间的比例不变。

【例 24-41】

```
M=[3 5;7 9;2 6];  
N=normr(M)
```

输出为：

```
N =  
    0.5145    0.8575  
    0.6139    0.7894  
    0.3162    0.9487
```

11) normc

应用 正则化矩阵的列。

格式 N=normc(M)

解析 normc 函数对其输入矩阵 M 进行正则化，函数返回的正则化矩阵 N 与原始矩阵维数相同，矩阵每一列元素的平方和变为 1，但各元素之间的比例不变。



【例 24-42】

```
M=[3 5;7 9;2 6];
N=normc(M)
```

输出为:

```
N =
    0.3810    0.4196
    0.8890    0.7553
    0.2540    0.5035
```

12) pnormc

应用 伪正则化矩阵的列。

格式 `N=pnormc(M,R)`

解析 `pnormr` 函数通过对输入矩阵 `M` 的每一列添加一个元素得到伪正则化矩阵 `N`, `N` 中每一列元素的平方和等于输入参数 `R` 的平方。

【例 24-43】

```
M=[3 5;7 9;2 8];
N=pnormc(M,6)
```

输出为:

```
N =
    3.0000    5.0000
    7.0000    9.0000
    2.0000    8.0000
    0 + 5.0990i    0 + 11.5758i
```

13) quant

应用 将实数量化为某一数值的整数倍。

格式 `quant(X,Q)`

解析 `quant` 函数将矩阵 `X` 中的每个元素离散化为量化因子 `Q` 的最近整数倍。

【例 24-44】

```
X=[1.6372 5.3578 -1.7892];
Y=quant(X,0.2)
```

输出为:

```
Y =
    1.6000    5.4000   -1.8000
```

14) seq2con

应用 将串行向量转变为并行向量。

格式 `b=seq2con(s)`

解析 在神经网络工具箱中，矩阵用于存储神经网络的并行输入向量，单元数组用于存储网络的串行输入向量。seq2con 函数可以实现网络串行输入向量到并行输入向量的转换。

函数输入 s 为 $N \times TS$ 维单元数组， s 中的每一个元素是由 M 列向量构成的矩阵。函数返回 $N \times 1$ 维的单元数组 b ， b 中的每一个元素是由 $M \times TS$ 列并行输入向量构成的矩阵。

【例 24-45】

```
p1={1 5 9};
p2=seq2con(p1)
```

输出为：

```
p2 =
    [1x3 double]
```

由上述结果可知，单元数组 $p2$ 中的唯一元素是一个 1×3 维矩阵。

```
p2{1}
ans =
     1     5     9
```

15) sumsqr

应用 求矩阵的平方和。

格式 sumsqr(m)

解析 sumsqr 函数用来求取函数输入矩阵 m 中各元素的平方和。

【例 24-46】

```
s=sumsqr([3 7;2 8])
s =
    126
```

24.7 绘图函数

绘图函数如表 24-7 所示。

表 24-7 绘图函数

函数名称	功 能	函数名称	功 能
hintonw	绘制权值矩阵的 Hinton 图	plotpc	在感知器输入向量图上绘制分类线
hintonwb	绘制权值矩阵和阈值向量的 Hinton 图	plotperf	绘制网络训练过程中的性能变化曲线
plotbr	绘制网络采用 Bayesian 正则化算法训练时的性能变化曲线	plotsom	绘制自组织映射网络图
		plotv	绘制起自原点的向量
plotes	绘制单输入神经元的误差曲面	plotvec	用不同颜色绘制向量
plotpv	根据目标向量绘制感知器的输入向量	plotep	在单输入神经元的误差曲面上绘制权值、阈值和相应误差点的位置



1) plotpc

应用 分界线绘制函数。

格式 `plotpc(W, b)`

`plotpc(W, b, h)`

解析 该函数用于在感知器向量图中绘制分界线，其参数含义如下。

W: $S \times R$ 维的加权矩阵 (R 必须小于等于 3);

b: $S \times 1$ 维的阈值向量;

h: 最后画线的控制权;

`plotpc(W, b)`: 返回的是对所绘制分界线的控制权;

`plotpc(W, b, h)`: 用于绘制新线之间检查最新绘制的分界线。

2) plotpv

应用 输入/目标向量绘制函数。

格式 `plotpv(p, t)`

`plotpv(p, t, v)`

解析 该函数用于绘制感知器的输入向量和目标向量，其参数含义如下。

p: Q 组 R 维的输入向量;

t: Q 组 S 维的双目标向量;

v=[x_{\min} x_{\max} y_{\min} y_{\max}]: 图形的最大值，绘制工作必须位于 v 所限定的范围中;

`plotpv(p, t)`: 以 t 为标尺，绘制 p 的列向量;

`plotpv(p, t, v)`: 在 v 的范围中绘制 p 的列向量。

【例 24-47】 本例主要目的在于演示函数 `plotpc` 和 `plotpv` 的应用。

假定已给出了某感知器的输入变量 p 和目标变量 t ，绘制其曲线；然后给定感知器的权值和阈值，绘制其分界线。

MATLAB 代码如下。

```
p=[0 0 1 1;0 1 0 1];
t=[0 0 0 1];
%绘制输入向量和目标向量
plotpv(p,t)
net=newp(minmax(p),1); %创建一个感知器网络
%设定权值
net.iw{1,1}=[-1.2 -0.5];
net.b{1}=1; %设定阈值
plotpc(net.iw{1,1},net.b{1})
```

运行结果如图 24-16 所示。

3) hintonw

应用 绘制权值矩阵的 Hinton 图。

格式 `hintonw(W, maxw, minw)`

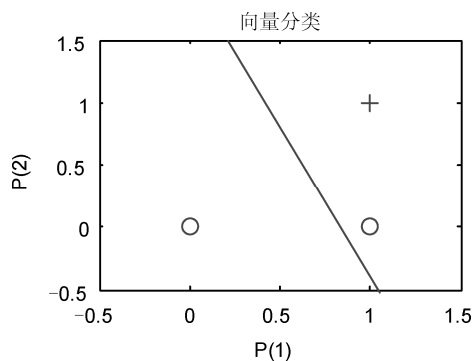


图 24-16 函数 plotpc 和 plotpv 演示运行结果

解析 `hintonw` 函数以方块图的形式表示权值矩阵，图中各方块的面积正比于权值矩阵中相应元素的大小，方块的颜色表示该元素的符号。其中，深色表示负权值，浅色表示正权值。该函数的输入为权值矩阵 W 、最大权值 `maxw` 和最小权值 `minw`。

【例 24-48】根据下面的权值矩阵绘制 Hinton 图。

```
W=[0.9631 -0.2345 0.0235 -0.0123;
   -0.3245 0.8963 -0.6348 -0.3517;
   0.5681 0.3964 -0.6321 0.8527]
hintonw(W)
```

运行结果如图 24-17 所示。

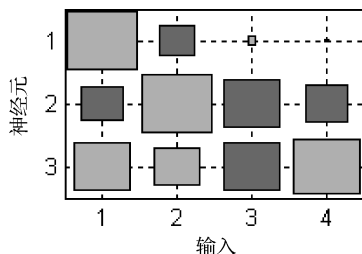


图 24-17 权值的 Hinton 图

4) `hintonwb`

应用 绘制权值矩阵和阈值向量的 Hinton 图。

格式 `hintonwb(W, b, maxw, minw)`

解析 `hintonwb` 函数以方块图的形式表示权值矩阵和阈值向量，图中各方块的面积正比于权值矩阵或阈值向量中相应元素的大小，方块的颜色表示该元素的符号，其中，深色表示负值，浅色表示正值。该函数的输入为权值矩阵 W 、阈值向量 b 、最大权值 `maxw` 和最小权值 `minw`。

【例 24-49】根据下面的权值矩阵和阈值向量绘制 Hinton 图。

```
W=[0.9631 -0.2345 0.0235 -0.0123;
   -0.3245 0.8963 -0.6348 -0.3517;
```



```
0.5681 0.3964 -0.6321 0.8527];
b=[0.4536;-0.6321;0.1235];
hintonwb(W,b)
```

运行结果如图 24-18 所示。

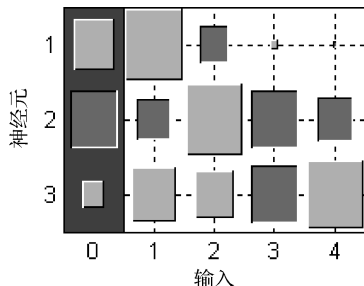


图 24-18 权值和阈值的 Hinton 图

5) plotbr

应用 绘制网络采用 Bayesian 正则化算法训练时的性能变化曲线。

格式 plotbr (TR, name, epoch)

解析 当网络的训练函数为 trainbr 时，可以利用 plotbr 函数绘制训练过程中网络性能的变化曲线，函数绘制的曲线包括网络输出的方差和、各权值参数的平方和以及网络中有效参数的个数。函数的输入 TR 为训练函数产生的训练记录；name 为训练函数名称，默认值为空字符串；epoch 为要显示的训练次数，其默认值为训练记录的长度。

【例 24-50】 利用下面一组代码演示 plotbr 函数记录训练过程曲线。

MATLAB 代码如下。

```
p=[-1:0.1:1];
t=sin(2*3.14159*p)+0.15*randn(size(p));
net=newff([-1 1],[20,1],{'tansig','purelin'},'trainbr');
[net,tr]=train(net,p,t);
plotbr(tr);
```

运行结果如图 24-19 所示。

6) plotes

应用 用于绘制一个单独神经元的误差曲面。

格式 plotes (wv, bv, es, v)

解析 plotes 函数可用来绘制单输入神经元的误差曲面和等高线。wv 为神经元可能权值范围向量；bv 为神经元可能的阈值范围向量；es 是由 errsrf 函数计算出的误差矩阵，该矩阵给出了在权值范围向量和阈值范围向量各组合点上的神经元输出误差；v 给出了三维图形的视点，默认值为[-37.5, 30]。

示例参看例 24-19 所示。

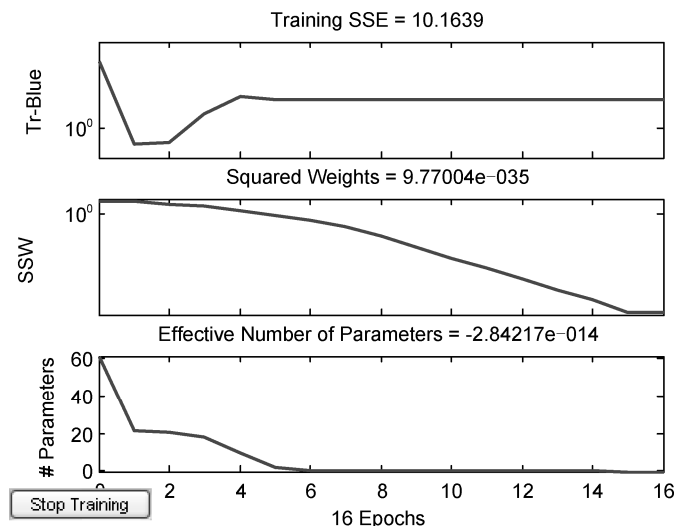


图 24-19 Bayesian 正则化训练时的网络性能变化曲线

7) plotep

应用 在单输入神经元的误差曲面上绘制指定权值、阈值及相应误差的位置。

格式 `H=plotep(W,B,E)`

`H=plotep(W,B,E,H)`

解析 `plotep` 函数可以在 `plotes` 函数所绘的图形上标识指定权值、阈值和相应误差点的位置。在函数的输入中, `W` 为权值, `B` 为阈值, `E` 为误差, `plotep` 函数将在等高线图上绘出对应于 `W` 和 `B` 的点, 在误差曲面上绘出对应于 `E` 的点, 同时函数返回的句柄 `H` 保存了本次函数所绘各点的信息。调用 `H=plotep(W,B,E,H)` 利用上次调用函数时返回的句柄 `H`, 在绘制新点前删除旧点。

8) plotperf

应用 绘制网络训练过程中的性能变化曲线。

格式 `plotperf(TR,goal,name,epoch)`

解析 `plotperf` 函数用于绘制网络在训练时的性能变化曲线, 如果训练中有验证步骤和测试步骤, 本函数还将绘制验证性能和测试性能的变化曲线。函数的输入 `TR` 为训练记录; `goal` 为网络性能目标, 默认值为 `NaN`; `name` 为训练函数的名称; `epoch` 为要显示的训练次数, 其默认值为训练记录的长度。

【例 24-51】有一个输入向量 `P` 和目标向量 `T`, 分别有 8 个向量, 由此导出一组确认样本如下。

```
P=1:8;
T=sin(P);
VV.P=P;
VV.T=T+rand(1,8)*0.1;
```

创建一个 BP 网络, 并进行训练, 找出 `P` 和 `T` 之间的非线性关系, 并利用确认样本对网



络进行检验。

```
net=newff(minmax(P),[4 1],{'tansig','tansig'});
[net,tr]=train(net,P,T,[],[],VV);
plotperf(tr);
```

网络的训练误差曲线如图 24-20 所示，训练结果为：

```
TRAINLM, Epoch 0/100, MSE 1.32542/0, Gradient 4.61101/1e-010
TRAINLM, Epoch 14/100, MSE 0.374589/0, Gradient 0.00939335/1e-010
TRAINLM, Validation stop.
```

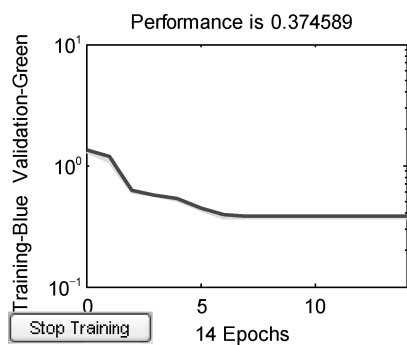


图 24-20 网络的训练记录

9) plotsom

应用 绘制自组织映射网络图。

格式 plotsom (pos)

plotsom (W, D, nd)

解析 plotsom 函数用于绘制自组织映射网络图。在函数调用 plotsom (pos)中，pos 是网络中各神经元在物理空间分布的位置坐标矩阵；函数返回神经元物理分布的拓扑图，图中每两个间距小于 1 的神经元以直线连接。在函数调用 plotsom (W, D, nd)中，W 为神经元权值矩阵；D 为根据神经元位置坐标计算出来的间距矩阵；nd 为领域半径，默认值为 1；函数返回神经元权值的分布图，图中每两个间距小于 nd 的神经元以直线连接。

【例 24-52】 构造一个二输入八神经元的自组织映射网络层，网络的拓扑结构为长方形，神经元间距采用 linkdist 计算方法并随机产生权值矩阵 W。

MATLAB 代码如下：

```
pos=gridtop(2,4);
W=rand(8,2);
%绘制神经元拓扑结构图和权值向量;
subplot(1,2,1)
plotsom(pos);
D=linkdist(pos);
subplot(1,2,2)
plotsom(W,D)
```

运行结果如图 24-21 所示。

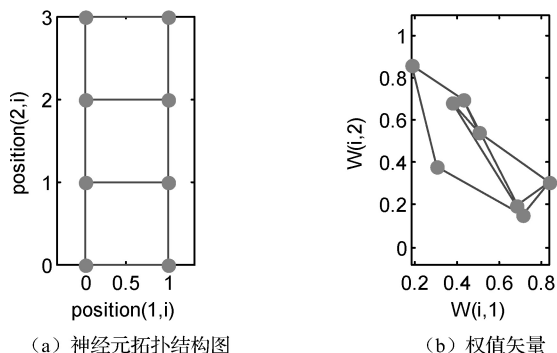


图 24-21 自组织映射网络中的神经元拓扑结构图和权值向量

10) plotv

应用 绘制起自原点的向量。

格式 `plotv(M, t)`

解析 `plotv` 函数用于绘制向量图，函数的输入 **M** 为向量矩阵；**t** 指定绘图时向量线的形状，默认值为 '-'；函数将绘制 **M** 中的每一个列向量。

【例 24-53】 绘制下面的向量。

```
w=[-0.4 0.7;-0.5 0.2];
plotv(w,'ro')
```

运行结果如图 24-22 所示。

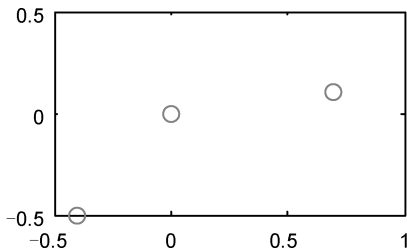


图 24-22 向量图

11) plotvec

应用 用不同颜色绘制向量。

格式 `plotvec(X, C, m)`

解析 `plotvec` 函数利用不同颜色绘制向量。函数的输入 **X** 为向量矩阵；**C** 是表示颜色坐标的行向量；**m** 指定绘图时向量的标识符号。函数将绘制 **X** 中的每一个列向量，每个向量的颜色由向量 **C** 中的对应元素确定。

【例 24-54】 针对一组输入向量，设计一个 LVQ 神经网络，经过训练后，能对给定数据进行模式识别。

MATLAB 代码如下：

```
P=[-6 -4 -2 0 0 1 2 4 6;0 2 -2 1 2 -2 1 2 -2 0];
C=[1 1 1 2 2 2 2 1 1 1];
T=ind2vec(C);
plotvec(P,C,'+b');
axis([-6 6 -3 3]);
net=newlvq(minmax(P),5,[0.6 0.4]);
net.trainParam.epochs=85;
net=train(net,P,T);
p=[0 1;0.2 0];
y=sim(net,p);
vc=vec2ind(y)
```

运行效果如图 24-23 和图 24-24 所示，在命令窗口中得到下面的结果。

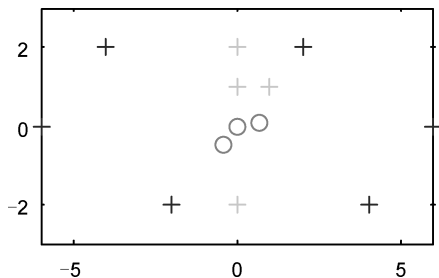


图 24-23 输入样本数据分布图

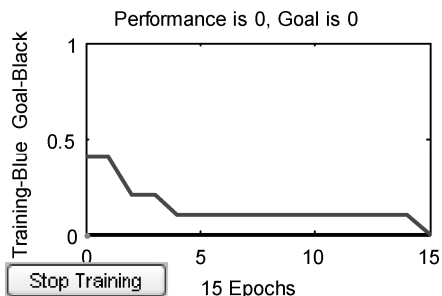


图 24-24 训练误差曲线

```
%用 TRAINR 作为训练函数，最大训练次数为 85 次
RAINR, Epoch 0/85
TRAINR, Epoch 15/85
TRAINR, Performance goal met.
%对给定数据，一个归于第 2 类，一个归于第 1 类
vc =
    2     1
```

24.8 数据预处理和后处理函数

网络训练前、后数据的预处理和后处理函数如表 24-8 所示。

表 24-8 数据预处理和后处理函数

函数名称	功 能
premnmx	把数据归一化到-1 和 1 之间
postmnmx	恢复被函数 premnmx 归一化的数据

续表

函数名称	功 能
prestd	把数据归一化为单位方差和零均值
poststd	恢复被函数 prestd 归一化的数据
postreg	利用线性回归分析对神经网络的仿真结果进行后处理
prepca	对输入数据进行主元分析
trapca	利用预先计算的主元分析矩阵对数据进行变换
trastd	利用预先计算的均值和方差对数据进行变换
tramnmx	利用预先计算的最大值和最小值对数据进行变换

1) premnmx

应用 把数据归一化到-1 和 1 之间。

格式 $[P_n, \min_p, \max_p, T_n, \min_t, \max_t] = \text{premnmx}(P, T)$

$[P_n, \min_p, \max_p] = \text{premnmx}(P)$

解析 premnmx 函数用于对网络的输入数据或目标数据进行归一化，归一化后的数据将分布在 $[-1, 1]$ 区间内。归一化公式为：

$$P_n = 2 * (P - \min_p) / (\max_p - \min_p) - 1$$

$$T_n = 2 * (t - \min_t) / (\max_t - \min_t) - 1$$

其中，P 为原始输入数据，maxp 和 minp 分别是 P 中的最大值和最小值，Pn 为归一化后的输入数据。T 是原始目标数据，maxt 和 mint 分别是 T 中的最大值和最小值，Tn 是归一化后的目标数据。

函数调用 $[P_n, \min_p, \max_p, T_n, \min_t, \max_t] = \text{premnmx}(P, T)$ 可以把网络输入数据 P 和目标数据 T 归一化为 Pn 和 Tn，同时返回 P 中的最小值 minp 和最大值 maxp，以及 T 中的最小值 mint 和最大值 maxt，也可以调用 $[P_n, \min_p, \max_p] = \text{premnmx}(P)$ 形式只对输入数据 P 进行归一化。

2) postmnmx

应用 恢复被函数 premnmx 归一化的数据。

格式 $[P, T] = \text{postmnmx}(P_n, \min_p, \max_p, T_n, \min_t, \max_t)$

$[P] = \text{postmnmx}(P_n, \min_p, \max_p)$

解析 该函数用于恢复被函数 premnmx 归一化的数据，变换公式为：

$$p = 0.5 * (P_n + 1) * (\max_p - \min_p) + \min_p$$

其中，P 为原始数据，maxp 和 minp 分别是 P 中的最大值和最小值，Pn 为归一化后的数据。

同理

$$T = 0.5 * (T_n + 1) * (\max_t - \min_t) + \min_t$$

函数调用 $[P, T] = \text{postmnmx}(P_n, \min_p, \max_p, T_n, \min_t, \max_t)$ 可以在已知 P 中的最小值 minp 和最大值 maxp，以及 T 中的最小值 mint 和最大值 maxt 的前提下，把归一化的网络输入数据 Pn 和目标数据 Tn 恢复为原始数据 P 和 T。也可以利用函数 $[P] = \text{postmnmx}(P_n, \min_p, \max_p)$ 形式只恢复归一化输入数据 P。



【例 24-55】把下列数据归一化到[-1, 1]区间内并恢复其原始数据。

```
P=[-9 -6 -4.5 -1.5 0];
T=[0 8.8 -11 -10 2];
%首先对数据进行归一化
[Pn,minp,maxp,Tn,mint,maxt]=premnmx(P,T)
Pn =
    -1.0000    -0.3333         0     0.6667     1.0000
minp =
    -9
maxp =
     0
Tn =
     0.1111     1.0000    -1.0000    -0.8990     0.3131
mint =
    -11
maxt =
     8.8000
%恢复原始数据
[P,T]=postmnmx(Pn,minp,maxp,Tn,mint,maxt)
P =
    -9.0000    -6.0000    -4.5000    -1.5000         0
T =
         0     8.8000    -11.0000    -10.0000     2.0000
```

3) prestd

应用 把数据归一化为单位方差和零均值。

格式 $[Pn, \text{meanp}, \text{stdp}, Tn, \text{meant}, \text{stdt}] = \text{prestd}(P, T)$

$[Pn, \text{meanp}, \text{stdp}] = \text{prestd}(P)$

解析 prestd 函数用于对网络的输入数据或目标数据进行归一化, 归一化后的数据将具有零均值和单位方差。归一化公式为

$$Pn = (P - \text{meanp}) / \text{stdp}$$

其中, P 和 Pn 分别为归一化前、后的输入数据, meanp 和 stdp 分别为原始数据 P 的均值和方差。同理

$$Tn = (T - \text{meant}) / \text{stdt}$$

其中, T 和 Tn 分别是归一化前、后的目标数据, meant 是原始数据 T 的均值, stdt 是其方差。

函数调用 $[Pn, \text{meanp}, \text{stdp}, Tn, \text{meant}, \text{stdt}] = \text{prestd}(P, T)$ 可以把网络的输入数据 P 和目标数据 T 归一化为 Pn 和 Tn, 同时返回 P 的均值 meanp 和方差 stdp, 以及 T 的均值 meant 和方差 stdt, 也可以调用 $[Pn, \text{meanp}, \text{stdp}] = \text{prestd}(P)$ 只对输入数据 P 进行归一化。



4) poststd

应用 恢复被函数 prestd 归一化的数据。

格式 $[P, T] = \text{poststd}(P_n, \text{meanp}, \text{stdp}, T_n, \text{meant}, \text{stdt})$

$[P] = \text{poststd}(P_n, \text{meanp}, \text{stdp})$

解析 poststd 函数用于恢复被函数 prestd 归一化的函数，变换公式为

$$P = P_n * \text{stdp} + \text{meanp}$$

其中，P 和 P_n 分别为归一化前、后的数据，meanp 和 stdp 分别为原始数据 P 的均值和方差。同理

$$T = T_n * \text{stdt} + \text{meant}$$

函数调用 $[P, T] = \text{poststd}(P_n, \text{meanp}, \text{stdp}, T_n, \text{meant}, \text{stdt})$ 可以把归一化数据 P_n 和 T_n 恢复为原始数据 P 和 T，同时需要已知 P 的均值 meanp 和方差 stdp，以及 T 的均值 meant 和方差 stdt，也可以调用 $[P] = \text{poststd}(P_n, \text{meanp}, \text{stdp})$ 只恢复归一化输入数据 P。

【例 24-56】 把下列数据归一化为具有零均值和单位方差的数据序列并恢复其原始数据。

```
P=[-9 -6 -4.5 -1.5 0];
T=[0 8.8 -11 -10 2];
%首先对数据进行归一化
[Pn,meanp,stdp,Tn,meant,stdt]=prestd(P,T)
Pn =
    -1.3403    -0.5026    -0.0838     0.7539     1.1728
meanp =
    -4.2000
stdp =
     3.5812
Tn =
     0.2431     1.2919    -1.0678    -0.9486     0.4815
meant =
    -2.0400
stdt =
     8.3909
%恢复原始数据
[P,T]=poststd(Pn,meanp,stdp,Tn,meant,stdt)
P =
    -9.0000    -6.0000    -4.5000    -1.5000         0
T =
         0     8.8000   -11.0000   -10.0000     2.0000
```

5) postreg

应用 利用线性回归分析对神经网络的仿真结果进行后处理。

格式 $[M, B, R] = \text{postreg}(A, T)$



解析 该函数对网络的仿真输出和目标向量进行线性回归分析，并得到目标向量对网络输出的相关系数，从而可以作为检验网络性能的参数。

函数的输入分别为网络输出向量 A 和目标向量 T ，函数返回线性拟合直线的斜率 M ，截距系数 B 以及输出向量 A 和目标向量 T 之间的相关系数 R 。当 R 为 1 时，输出和目标向量之间的相关性最好。

【例 24-57】 利用级联前向网络对下列样本数据进行学习，并对网络的实际输出向量和目标向量进行线性回归分析。

MATLAB 代码如下。

```
P=[0.96 -0.78 -0.53 0.74 0.31];
net=newff(minmax(P),[5 1],{'tansig' 'purelin'},'trainlm');
T=[-0.03 3.5 -0.9 0.65 3.0];
net=train(net,P,T);
A=sim(net,P);
```

运行结果如图 24-25 所示。

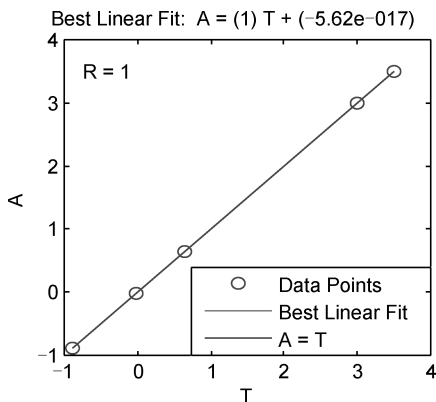


图 24-25 线性回归分析结果

6) prepca

应用 对输入数据进行主元分析。

格式 [Ptrans, TransMat]=prepca(P, min_frac)

解析 prepca 函数用来对输入数据矩阵 P 进行主元分析变换，该变换可以消除各输入向量间的相关性。在变换后的矩阵中，部分向量占用原始数据矩阵的大部分能量，保存了原始数据的大部分信息，因此变换后应予以保留；而有些向量只占有原始数据的小部分能量，因此可以在变换后省略。

函数的输入 P 为原始数据矩阵；min_frac 为最小能量比例系数，当变换后矩阵中某一向量的能量在原始数据总能量中所占的比例小于该系数时，这一向量将被省略。函数返回经过主元分析的矩阵 Ptrans 和变换时使用的矩阵 TransMat。在该函数算法中都假定原始数据具有零均值，因此要首先利用函数 prestd 对数据进行归一化。

【例 24-58】 对以下数据进行主元分析。



```

p=[-1.5 -0.58 0.21 -0.96;-2.2 -0.87 0.34 -1.4;-2.5 -0.38 0.44 -1.06];
pn=prestd(p);
[ptrans,transMat]=prepca(pn,0.01)
ptrans =
    2.0218    -0.4191    -2.0912     0.4885
   -0.1787     0.1927    -0.1740     0.1601
transMat =
   -0.5797   -0.5791   -0.5732
   -0.3770   -0.4330     0.8187

```

7) trastd

应用 利用预先计算的均值和方差对数据进行变换。

格式 `[Pn]=trastd(P, meanp, stdp)`

解析 本函数利用 `prestd` 函数对样本数据进行归一化时得到的均值和方差新的输入数据进行变换。计算公式为：

$$P=Pn*stdp+meanp$$

式中的 **P** 和 **Pn** 分别为变换前、后的输入数据，**meanp** 和 **stdp** 分别为 `prestd` 函数找到的均值和方差。如果网络训练时所用的是归一化样本数据，那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理，这时就要用到 `trastd` 函数。

【例 24-59】 利用级联前向网络对下列归一化样本数据进行学习，并用新的输入数据检验网络。

```

P=[-1.92 2.37 -1.48 0.86 3.12];
T=[-0.39 3.6 -0.95 0.75 3.2];
[Pn,minp,maxp,Tn,mint,maxt]=prestd(P,T);
net=newff(minmax(P),[5 1],{'tansig' 'purelin'},'trainlm');
net=train(net,Pn,Tn);
P2=[2 -2];
P2n=trastd(P2,minp,maxp);
A2n=sim(net,P2n);
A2=poststd(A2n,mint,maxt)

```

运行显示如图 24-26 所示的训练过程，窗口显示如下。

```

A2 =
    3.5470   -0.2676

```

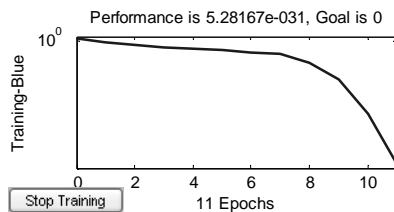


图 24-26 训练过程



8) trapca

应用 利用预先计算的主元分析矩阵对数据进行变换。

格式 [Ptrans]=trapca(P, TransMat)

解析 本函数利用 prepca 函数对样本数据进行主元分析时得到的变换矩阵 TransMat 新的输入数据 P 进行变换。如果网络训练时所用的是经过主元分析的样本数据, 那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理, 这时就要用到 trapca 函数。

【例 24-60】 利用级联前向网络对下列经过主元分析的样本数据进行学习, 并用新的输入数据检验网络。

```
P=[-1.5 -0.58 0.21 -0.96 2.29;-2.2 -0.87 0.34 -1.4 -1.2];
T=[-0.39 3.6 -0.95 0.75 3.2];
[Pn,meanp,stdp,Tn,meant,stdt]=prestd(P,T);
[ptrans,transMat]=prepca(Pn,0.02);
net=newff(minmax(ptrans),[5 1],{'tansig' 'purelin'},'trainlm');
net=train(net,ptrans,Tn);
P2=[2.5 -1.8;-0.9 -2];
P2n=trastd(P2,meanp,stdp);
P2trans=trapca(P2n,transMat);
A2n=sim(net,P2trans);
A2=poststd(A2n,meant,stdt)
```

运行结果如下:

```
A2 =
    2.6182    -0.6545
```

9) tramnmx

应用 利用预先计算的最大值和最小值对数据进行变换。

格式 [Pn]=tramnmx(P, minp, maxp)

解析 本函数利用 premnmx 函数对样本数据进行归一化时得到的最小值和最大值对新的输入数据进行变换, 计算公式为:

$$P_n = 2 * (P - \min_p) / (\max_p - \min_p) - 1$$

式中的 P 和 Pn 分别为变换前、后的输入数据, maxp 和 minp 分别为 premnmx 函数找到的最大值和最小值。如果网络训练时所用的是归一化样本数据, 那么以后使用网络时所用的新输入数据也应该和样本数据一样接受相同的预处理, 这时就要用到 tramnmx 函数。

【例 24-61】 利用级联前向网络对下列归一化样本数据进行学习, 并用新的输入数据检验网络。

```
P=[-1.5 -0.58 0.21 -0.96 2.29];
T=[-0.39 3.6 -0.95 0.75 3.2];
[Pn,minp,maxp,Tn,mint,maxt]=premnmx(P,T);
net=newff(minmax(Pn),[5 1],{'tansig' 'purelin'},'trainlm');
net=train(net,Pn,Tn);
```



```
P2=[2 -2];  
P2n=trmnmx(P2,minp,maxp);  
A2n=sim(net,P2n);  
A2=postmnmx(A2n,mint,maxt)
```

运行窗口输出结果为:

```
A2 =  
    1.2015    -0.4051
```

第 25 章 神经网络的工程应用

25.1 线性神经网络在线性预测中的应用

【例 25-1】 利用 `adapt` 函数对自适应滤波网络进行训练，并用设计好的自适应滤波网络对某时变正弦信号进行自适应预测。

(1) 定义输入向量和目标向量。

待预测的时变正弦信号定义如下：

$$T = \begin{cases} \sin(4\pi k) & k \leq 4d \\ \sin(8\pi k) & 4s < k \leq 6s \end{cases}$$

可见，4s 以后正弦信号频率加倍。此外，信号的采样频率前 4s 每秒采样 20 次，4s 以后加倍。该时变正弦信号可以采用如下 MATLAB 语句生成：

```
t1=0:0.05:4;  
t2=4.04:0.024:6;  
t=[t1 t2];  
T=[sin(t1*4*pi) sin(t2*8*pi)];
```

为了便于训练，将信号转换成序列的形式：

```
T=con2seq(T);
```

令网络输入信号与目标序列相同

```
P=T;
```

(2) 网络设计。

在每一个采样时刻，我们采用该时刻之前的 5 个采样信号值作为网络的输入，网络的输出则为下一时刻信号的预测值。据此，线性自适应滤波网络可以按如图 25-1 所示的结构进行设计。

首先，采用 `newlin` 函数生成如图 25-1 所示的自适应滤波网络，即

```
lr=0.1;  
delays=[1 2 3 4 5];  
net=newlin(minmax(cat(2,P{:})),1,delays,lr);
```

其中，网络学习速率 `lr` 取 0.1。

然后，利用 `adapt` 函数对所生成的神经网络进行训练。

[net,Y,E]=adapt(net,P,T);

训练完成之后，即返回设计好的自适应滤波网络。

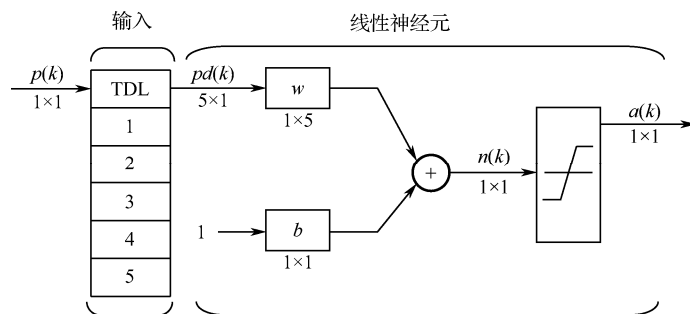


图 25-1 自适应滤波网络结构图

(3) 网络测试。

为了检测网络的预测效果，在网络训练完成之后，可以将网络预测输出与目标信号绘制在同一幅图中加以比较，如图 25-2 所示，其中虚线代表目标信号。

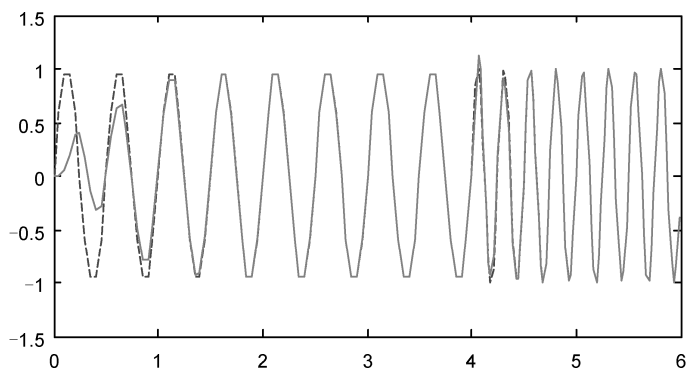


图 25-2 预测信号与目标信号曲线

由图 25-2 可见，在经过大约 1.5s 的自适应训练之后，网络几乎可以完全跟踪目标信号（曲线几乎完全重合）。在第 4s 时刻，由于目标信号的频率发生突变，所以网络的输出与目标信号曲线稍有偏差，但由于神经网络先前已经对与当前信号相似的正弦信号进行了学习，所以经过更短的训练时间就能再次精确地预测目标信号。

图 25-3 给出了网络的预测误差曲线。

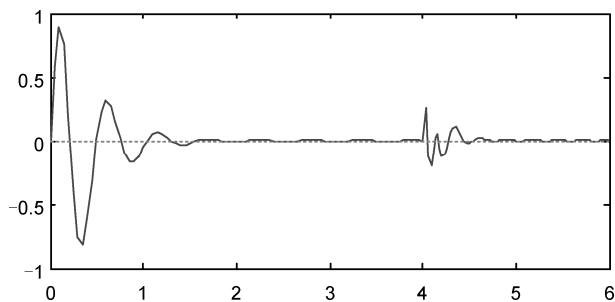


图 25-3 预测误差曲线



(4) 本例完整的 MATLAB 程序代码。

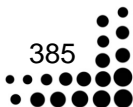
```
clear all
%定义输入向量和目标向量
t1=0:0.05:4;
t2=4.04:0.024:6;
t=[t1 t2];
T=[sin(t1*4*pi) sin(t2*8*pi)];
T=con2seq(T);
%目标信号
P=T;
%绘制待预测的目标信号曲线
plot(t,cat(2,T{:}))
%生成自适应滤波网络
%学习速率
lr=0.1;
delays=[1 2 3 4 5];
net=newlin(minmax(cat(2,P{:})),1,delays,lr);
%对网络进行训练
[net,Y,E]=adapt(net,P,T);
%绘制网络预测输出曲线
plot(t,cat(2,T{:}),'-',t,cat(2,Y{:}),'r')
clf
%绘制预测误差曲线
plot(t,cat(2,E{:}),[min(t) max(t)],[0 0],':r')
```

25.2 神经模糊控制在洗衣机中的应用

20 世纪 90 年代初期,日本松下公司推出了神经模糊控制全自动洗衣机。这种洗衣机能够自动判断衣物质地的软硬程度、衣量多少、脏污程度和性质等,应用神经模糊控制技术,自动生成模糊控制规则和隶属度函数,预设洗衣水位、水流强度和洗涤时间,在整个洗衣过程中实时调整这些参数,达到最佳的洗衣效果。

25.2.1 洗衣机的模糊控制

洗衣机的主要被控参量为洗涤时间和水流强度,而影响这一输出参量的主要因子是被洗物的浑浊程度和浑浊性质,后者可用浑浊度的变化率来描述。在洗涤过程中,油污的浑浊度变化率小,泥污的浑浊度变化率大。因此,浑浊度及其变化率可以作为控制系统的输入参量,而洗涤时间和水流强度可作为控制量,即系统的输出。实际上,洗衣过程中的这类输入和输出之间很难用一定的数学模型进行描述。系统运行过程中具有较大的不确定性,控制过程在





很大程度上依赖操作者的经验，这样一来，利用常规的方法进行控制难以奏效。但是，如果利用专家知识进行控制决策，往往容易实现优化控制，这就是在洗衣机中引入模糊控制技术的主要原因之一。

根据上述的模糊控制基本原理，可得出确定洗涤时间的模糊推理框图，如图 25-4 所示。其输入量为洗涤水的浑浊度及其变化率，输出量为洗涤时间。考虑到适当的控制性能需要和简化程序，定义输入量浑浊度的模糊词集为{清、较浊、浊、很浊}，定义浑浊度变换率的模糊词集为{零、小、中、大}，定义输出变量洗涤时间的模糊词集为{短、较短、标准、长}。描述输入/输出变量的词集都具有模糊特性，可以用模糊集合表示。因此，模糊概念的确定问题就直接转换为求取模糊集合的隶属函数问题。

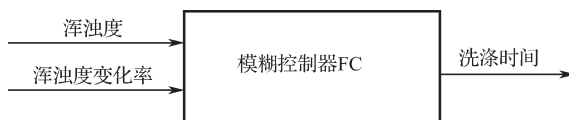


图 25-4 确定洗涤时间的模糊推理框图

通常定义一个模糊子集实际上就是确定模糊子集隶属函数的过程。将确定的隶属函数曲线离散化，就得到了有限个点上的隶属度，构成了一个相应的模糊子集。如图 25-5 所示，定义了隶属函数曲线表示论域 x 中元素 x 对模糊变量 A 的隶属程度。设定该隶属函数的论域 x 为：

$$x = (-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6)$$

则有：

$$\mu_A(2) = \mu_A(6) = 0.2, \quad \mu_A(3) = \mu_A(5) = 0.7, \quad \mu_A(4) = 1$$

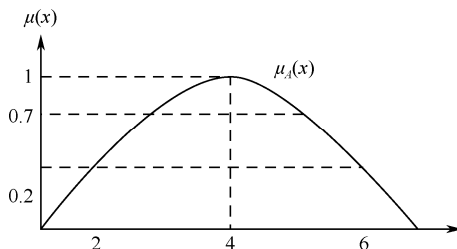


图 25-5 模糊变量 A 的隶属度函数

论域 x 中除了 2、3、4、5、6 外，各点的隶属度均为 0，那么模糊变量 A 的模糊子集为 $A=0.2/2+0.7/3+1/4+0.7/5+0.2/6$ 。

通过这个例子可以看出，在隶属函数的曲线确定后，就可以很容易地定义一个模糊变量的模糊子集。洗衣机模糊控制的输入和输出变量的隶属函数如图 25-6 所示，由此可以相继确定它们的模糊子集。

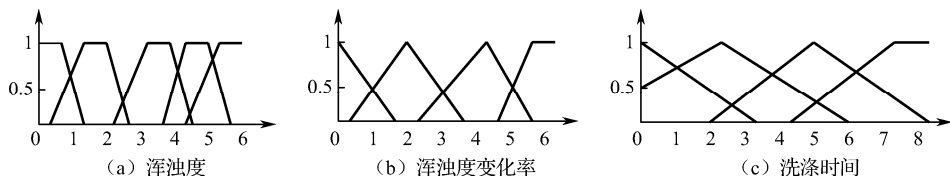


图 25-6 洗衣机模糊变量隶属函数

洗衣机的模糊控制规则可以归纳为 16 条，如表 25-1 所示。

表 25-1 洗衣机的模糊控制规则表

洗涤时间		浑浊度			
		清	较浊	浊	很浊
变化率	零	短	较短	标准	标准
	很小	标准	标准	标准	标准
	中	标准	长	长	长
	大	标准	标准	长	长

25.2.2 洗衣机的神经网络模糊控制器的设计

洗衣机模糊控制的控制部分框图如图 25-7 所示，模糊控制器如图中虚线所示。模糊控制的过程是这样的：首先洗衣机获取的浑浊度信息由传感器送到信息处理单元，分为浑浊度和浑浊度变化率，送入模糊控制器。对于输入的模糊量，需要将其转换成模糊变量，通过单元片机，利用查表法按照模糊推理法则做出决策，结果被认为是模糊变量，经过去模糊化单元处理，再由执行机构去修改洗涤时间，这样就完成了一次模糊控制算法过程。

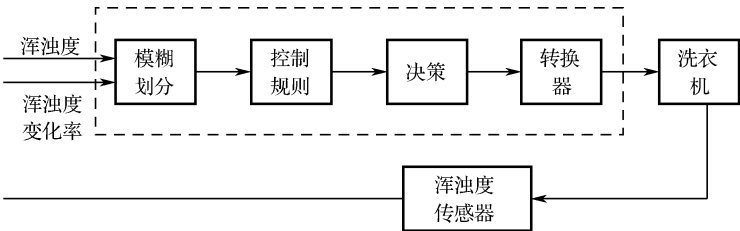


图 25-7 模糊控制的控制部分框图

一般的模糊控制洗衣机将“专家经验”通过模糊控制规则表现出来，运行中通过查表做出控制决策，这比需要操作者设定程序的电脑控制洗衣机前进了一大步。但是，这种洗衣机由于规则表需要占用大量的内存空间，查表反应速度慢，只能够按照已经编入的规则进行控制，因此不够理想。而把神经网络和模糊控制相结合，则能够解决这些问题。

洗衣机的神经网络模糊控制是利用离线训练好的网络，通过在线计算即可得到最佳输出。这种控制模式的反应速度快，而且神经网络又具有自学习功能和联想能力，对于未在训练中出现样本，也可以通过联系记忆的功能，做出控制决策，表现非常灵活。

洗衣机的神经网络模糊控制器的控制系统中含有多个神经模糊环节，下面仅介绍以浑浊度和浑浊度变化率为输入变量来确定洗涤时间的控制器。控制器的控制框图如图 25-8 所示，浑浊度神经网络结构如图 25-9 所示。

神经模糊控制器在输入/输出参量的选择及模糊论域和模糊子集的确定方面，与一般模糊控制器没有什么区别，只是在推理手段上引入了神经网络。令 $x_1 \sim x_7$ 为输入量浑浊度的模糊子集， $x_8 \sim x_{14}$ 为输入量浑浊度变化率的模糊子集， $y_1 \sim y_8$ 为输出控制量的模糊子集。从模

糊控制规则表 25-1 可以看出, 其有 16 条控制规则, 每条规则都是一对样本, 则共有 16 对样本。例如, 当浑浊度为“清”, 浑浊度变化率为“零”时, 洗涤时间应该为“短”, 这个样本可以表示为:

$$x = [1, 0.6, 0.1, 0, 0, 0, 0, 1.0, 0.5, 0, 0, 0, 0, 0]^T, \quad y = [1.0, 0.5, 0, 0, 0, 0, 0, 0]^T$$

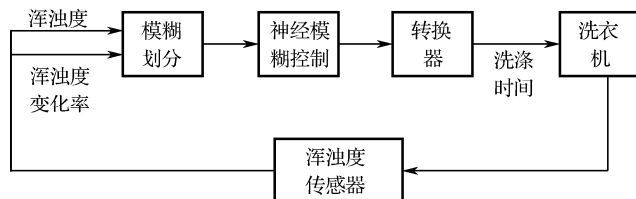


图 25-8 神经网络模糊控制器的控制框图

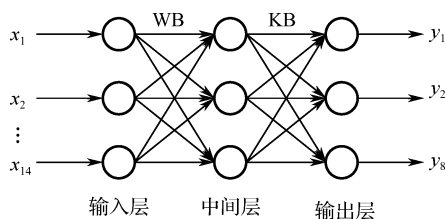


图 25-9 浑浊度神经网络结构

其中, x 中的各元素为对应的隶属函数, 即模糊子集的赋值。同理可列出其他 15 个样本对, 并将它们依次送入神经网络进行离线训练, 当训练结束后, 神经网络已经记忆了模糊控制规则, 使用时具有联想记忆功能。如表 25-2 所示为每一个输入参量的模糊量。

表 25-2 输入参量的模糊量

输入参量		模糊量
浑浊度	清	1 0.6 0.1 0 0 0 0
	较浊	0 0.6 0.6 0 0 0 0
	浊	0 0 0.6 1 0 0 0
	很浊	0 0 0 0 1 0.6 0
浑浊度变化率	零	1 0.5 0 0 0 0 0
	很小	0 0.5 1 0.4 0 0 0
	中	0 0 0 0.4 1 0.6 0
	大	0 0 0 0 0 0.8
洗涤时间	短	1 0.5 0 0 0 0 0
	较短	0.4 0.8 1 0.8 0.4 0.2 0 0
	标准	0 0 0 0.2 0.6 1 0.6 0.2
	长	0 0 0 0 0 0.2 0.5 0.8

根据模糊规则, 可以得到网络的训练样本 P 和 T, 完整的 MATLAB 代码如下。



```

P=[1 0.6 0.1 0 0 0 0 1 0.5 0 0 0 0 0;
    1 0.6 0.1 0 0 0 0 0 0.5 1 0.4 0 0 0;
    1 0.6 0.1 0 0 0 0 0 0 0 0.4 1 0.6 0;
    1 0.6 0.1 0 0 0 0 0 0 0 0 0 0 0.8;
    0 0.6 0.6 0 0 0 0 1 0.5 0 0 0 0 0;
    0 0.6 0.6 0 0 0 0 0 0.5 1 0.4 0 0 0;
    0 0.6 0.6 0 0 0 0 0 0 0 0.4 1 0.6 0;
    0 0.6 0.6 0 0 0 0 0 0 0 0 0 0 0.8;
    0 0 0.6 1 0 0 0 1 0.5 0 0 0 0 0;
    0 0 0.6 1 0 0 0 0 0.5 1 0.4 0 0 0;
    0 0 0.6 1 0 0 0 0 0 0.4 1 0.6 0;
    0 0 0.6 1 0 0 0 0 0 0 0 0 0 0.8;
    0 0 0 0 1 0.6 0 1 0.5 0 0 0 0 0;
    0 0 0 0 1 0.6 0 0 0.5 1 0.4 0 0 0;
    0 0 0 0 1 0.6 0 0 0 0.4 1 0.6 0;
    0 0 0 0 1 0.6 0 0 0 0 0 0 0 0.8];

T=[1 0.5 0 0 0 0 0 0;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0.4 0.8 1 0.8 0.4 0.2 0 0;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0 0.2 0.5 0.8;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0 0.2 0.5 0.8;
    0 0 0 0 0.2 0.5 0.8;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0.2 0.6 1 0.6 0.2;
    0 0 0 0 0.2 0.5 0.8;
    0 0 0 0 0.2 0.5 0.8];

%根据 Kolmogorov 定理, 由于输入层有 14 个节点, 所以中间层有 29 个节点
%中间层神经元的传递函数为 tansig
%输出层有 8 个节点, 其神经元传递函数为 logsig
%训练函数采用 traingdx
net=newff(minmax(P),[29,8],{'tansig','logsig'},'traingdx');
%训练步数为 1000 次
%训练目标误差为 0.001
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
net=train(net,P,T);

```



```
Y=sim(net,P);  
%求训练值在每一个点上的误差  
for i=1:16  
    x(i)=norm(Y(:,i));  
end  
plot(1:16,x);
```

网络的误差曲线如图 25-10 所示。由此可见，网络的最大误差不超过 0.2，说明网络性能是可以满足控制要求的。

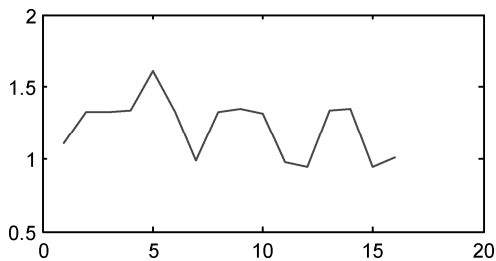


图 25-10 网络误差曲线

网络训练结果为：

```
TRAINGDX, Epoch 0/1000, MSE 0.28117/0.001, Gradient 0.082897/1e-006  
TRAINGDX, Epoch 25/1000, MSE 0.278678/0.001, Gradient 0.0830836/1e-006  
TRAINGDX, Epoch 50/1000, MSE 0.270858/0.001, Gradient 0.082787/1e-006  
TRAINGDX, Epoch 75/1000, MSE 0.246485/0.001, Gradient 0.0731321/1e-006  
TRAINGDX, Epoch 100/1000, MSE 0.201249/0.001, Gradient 0.0454135/1e-006  
TRAINGDX, Epoch 125/1000, MSE 0.127164/0.001, Gradient 0.0455586/1e-006  
TRAINGDX, Epoch 150/1000, MSE 0.0361097/0.001, Gradient 0.0142122/1e-006  
TRAINGDX, Epoch 175/1000, MSE 0.0125535/0.001, Gradient 0.00572951/1e-006  
TRAINGDX, Epoch 200/1000, MSE 0.00937949/0.001, Gradient 0.0020679/1e-006  
TRAINGDX, Epoch 225/1000, MSE 0.00223234/0.001, Gradient 0.00983431/1e-006  
TRAINGDX, Epoch 244/1000, MSE 0.000978229/0.001, Gradient 0.00205187/1e-006  
TRAINGDX, Performance goal met.
```

网络经过 244 次训练后，目标误差达到要求，结果如图 25-11 所示。

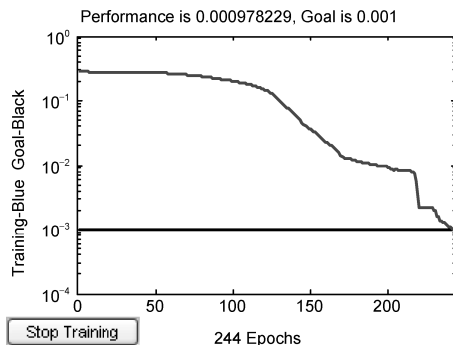


图 25-11 训练结果



25.3 模糊神经网络在配送中心选址中的应用

合理的物流配送中心选址能节省费用，加快货物的流通，增加物流企业的收益，因此，物流配送中心的选址决策对于整个物流系统的优化是个十分重要的问题。

1. 问题概述

长期以来，对于物流配送中心选址，人们提供了一种新的决策方法，如模糊综合评判法、AHP 层次分析法及结合层次法的模糊排序方法等。但这些方法也有一些缺点，利用模糊综合评判法，其指标权重难以确定；用专家打分法确定权重，人为因素又过重；利用层次分析法确定权重可以弱化人为因素，但是层次分析法要求指标的层次结构系统中的要素互相独立；否则就不能应用这种方法，而这些指标之间往往存在依赖关系，如地价和运输条件、政府政策和经营环境等。

模糊综合评价作为一种多属性的综合评价方法，其隶属函数权重有一定主观性，因此它的应用就有局限性。而 BP 算法则可以较客观地评价不同的方案，将模糊方法应用到 BP 算法的输入值中。综合两种算法的优势，通过网络学习得到选址的优化度，进而得到对于选址方案的一个较为准确的评价。

2. 设计

物流配送中心的选址通常是在一定的原则如降低成本原则、经济效益原则、提高客户服务水平原则等的指导之下，预先选择一些方案，然后再通过各种方法对这些方案进行比较，最终从中选出满意的一个或几个方案作为新中心的地址。影响配送中心选址的因素很多，可以根据物流学的原理，结合自身的实际情况，选择其中较重要的一些因素作为决策指标。这些因素大致可分为自然环境、交通运输条件、经营环境、候选地条件及公共设施几大类。指标选择的好坏对正确决策相当重要。选址主要考虑几个因素：客户的分布、供应商的分布、交通条件、用地条件、自然地理条件和环境条件等。

采用模糊方法计算影响因素的隶属度作为神经网络的输入，在 n 个影响因素中，既有定量因素也有定性因素，对不同的因素要用不同的方法来确定隶属度。对定量因素可以采用常见模糊分布来确定隶属函数，例如，模糊约束集合运输距离最小符合的模糊分布-降半梯形分布。对于定性因素而言，首先要用专家咨询法、专家评分法等方法进行量化，再利用相关的隶属函数确定隶属度。这里考虑的定量因素有候选地地价、运输距离和候选地面积。其中运输距离的隶属度通过上面介绍的方法确定，而另两种定量指标中，候选地地价与运输距离的隶属函数相同，模糊约束集合候选地面积适中符合模糊分布-正态分布，对应的隶属函数为：

$$u(x) = e^{-k(x-c)^2} \quad (k, c > 0)$$

而定性因素有客户分布、供应商分布、地质条件、通信条件、道路设施。定性因素首先对不同的情况做出分类描述，再将自然语言的评价分类转换为相应的隶属度，如表 25-3 所示。

表 25-3 定性指标的隶属函数

指 标	分 类 描 述	模 糊 约 束 集	隶 属 度
客户分布	集中, 一般, 分散	客户分布集中 A1	$A1=\{1, 0.5, 0\}$
供应商分布	集中, 一般, 分散	供应商分布集中 A2	$A2=\{1, 0.5, 0\}$
通信条件	很好, 好, 一般, 差, 很差	通信条件良好 A3	$A3=\{1, 0.8, 0.5, 0.2, 0\}$
地质条件	很好, 好, 一般, 差, 很差	地质条件良好 A4	$A4=\{1, 0.8, 0.5, 0.2, 0\}$

将指标隶属度输入网络中, 将专家值作为网络的期望输出。表 25-4 所示为经过量化和模糊处理, 并求出隶属度的实验训练数据。表 25-5 所示为经过初步筛选后的实验数据及其方案的评价结果和排序。

表 25-4 输入样本

方案序号	地质条件	客户分布	候选地地价	供应商分布	运输距离	通信条件	候选地面积	道路设施	专家评价结果
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	0.80	0.87	0.89	0.82	0.78	0.80	0.75	0.33	0.79
3	0.67	0.93	0.22	0.75	1.00	0.80	0.49	0.66	0.74
4	0.92	0.80	0.89	0.92	0.89	0.80	1.00	1.00	0.81
5	0.87	0.93	1.00	1.00	1.00	1.00	1.00	1.00	0.96
6	0.80	0.72	0.89	0.82	0.89	0.80	0.75	1.00	0.83
7	0.67	0.72	0.67	0.66	0.67	0.60	0.49	0.66	0.69
8	0.72	0.80	0.78	0.75	0.78	0.80	0.75	0.66	0.75
9	0.60	0.60	0.56	0.58	0.56	0.60	0.49	0.66	0.58
10	0.47	0.47	0.44	0.41	0.44	0.40	0.49	0.35	0.51

表 25-5 测试样本及测试结果

方案序号	地质条件	客户分布	候选地地价	供应商分布	运输距离	通信条件	候选地面积	道路设施	评价结果	排序
11	0.40	0.40	0.33	0.33	0.33	0.40	0.49	0.35	0.48	4
12	0.08	0.93	0.56	0.92	0.89	0.60	0.24	0.33	0.81	2
13	0.67	0.60	0.89	0.82	1.00	0.80	0.75	0.29	0.97	1
14	0.32	0.40	0.67	0.33	0.33	0.80	0.75	0.35	0.54	3
15	0.87	0.72	0.89	0.92	0.89	0.40	0.49	0.29	0.47	5

根据模糊规则, 可得到网络的训练样本 P 和 T, 完整的 MATLAB 代码如下。

```
P=[1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00;
    0.80 0.87 0.89 0.82 0.78 0.80 0.75 0.33;
    0.67 0.93 0.22 0.75 1.00 0.80 0.49 0.66;
    0.92 0.80 0.89 0.92 0.89 0.80 1.00 1.00;
```




```

0.87 0.93 1.00 1.00 1.00 1.00 1.00 1.00;
0.80 0.72 0.89 0.82 0.89 0.80 0.75 1.00;
0.67 0.72 0.67 0.66 0.67 0.60 0.49 0.66;
0.72 0.80 0.78 0.75 0.78 0.80 0.75 0.66;
0.60 0.60 0.56 0.58 0.56 0.60 0.49 0.66;
0.47 0.47 0.44 0.41 0.44 0.40 0.49 0.35];
T=[1.00 0.79 0.74 0.81 0.96 0.83 0.69 0.75 0.58 0.51];
%根据 Kolmogorov 定理, 由于输入层有 8 个节点, 所以中间层有 17 个节点
%中间层神经元的传递函数为 tansig
%输出层有 1 个节点, 其神经元传递函数为 logsig
%训练函数采用 trainlm
net=newff(minmax(P),[17,1],{'tansig','logsig'},'trainlm');
%训练步数为 1000 次
%训练目标误差为 0.001
net.trainParam.epochs=1000;
net.trainParam.goal=0.001;
net=train(net,P,T);
Y=sim(net,P);

```

网络的训练结果为:

```

TRAINLM, Epoch 0/1000, MSE 0.05559/0.001, Gradient 1.4427/1e-010
TRAINLM, Epoch 3/1000, MSE 0.000339403/0.001, Gradient 0.0831561/1e-010
TRAINLM, Performance goal met.

```

网络经过 3 次训练后, 目标误差达到要求, 结果如图 25-12 所示。

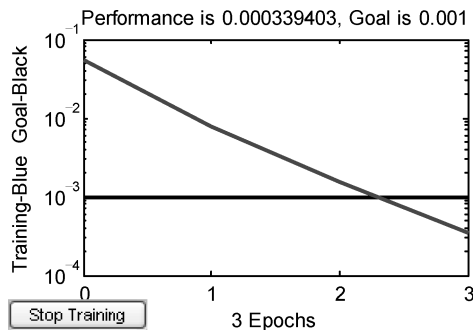


图 25-12 训练结果

预测过程代码为:

```

P_test=[0.40 0.40 0.33 0.33 0.33 0.40 0.49 0.35;
0.08 0.93 0.56 0.92 0.89 0.60 0.24 0.33;
0.67 0.60 0.89 0.82 1.00 0.80 0.75 0.29;
0.32 0.40 0.67 0.33 0.33 0.80 0.75 0.35;
0.87 0.72 0.89 0.92 0.89 0.40 0.49 0.29];
Y=sim(net,P_test)

```



输出结果为:

$Y = 0.4641 \quad 0.1850 \quad 0.3676 \quad 0.5007 \quad 0.6614$

从以上计算结果可以看出, 方案 13 最优, 方案 15 最差, 这也与各方案的指标因素特点基本一致, 根据此结果可进行配送中心选址的决策。

25.4 Elman 神经网络在信号检测中的应用

【例 25-2】 利用 Elman 网络对信号幅度进行检测。

1) 问题描述

待检测的信号是幅度变化的正弦信号, 可以通过如下的语句生成:

```
P1=sin(1:20);
P2=sin(1:20)*2;
P=[P1 P2 P1 P2];
```

该信号由幅度分别为 1 和 2 的正弦信号交替变化构成, 图 25-13 中所绘的虚线为待测的正弦信号。本例中要利用 Elman 网络对信号的幅度进行检测, 希望网络能够正确地输出时变信号的幅度值, 因此网络的目标输出为:

```
T1=ones(1,20);
T2=ones(1,20)*2;
T=[T1 T2 T1 T2];
```

目标输出曲线如图 25-13 中所绘的短画线。网络的输入和目标输出应采用串行序列格式:

```
Pseq=con2seq(P);
Tseq=con2seq(T);
```

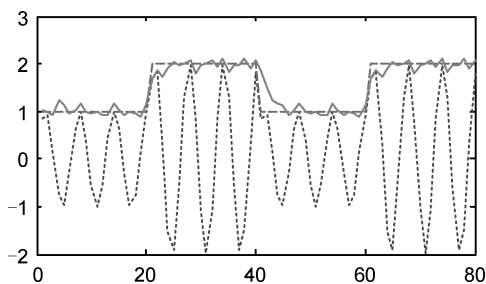


图 25-13 Elman 网络的信号幅度检测性能

2) 网络设计

建立两层神经元构成的 Elman 网络, 网络的隐层和输出层分别含有 10 个和 1 个神经元, 传递函数分别为 tansig 函数和纯线性函数, 网络训练函数为 traingdx 函数。

```
net=newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx')
```



3) 网络训练

设置好网络的训练参数后,就可以对网络进行训练了。

```
net.trainParam.epochs=1000;
net.trainParam.show=20;
net.trainParam.goal=0.01;
net.performFcn='sse';
[net,tr]=train(net,Pseq,Tseq);
```

训练结果如图 25-14 所示。

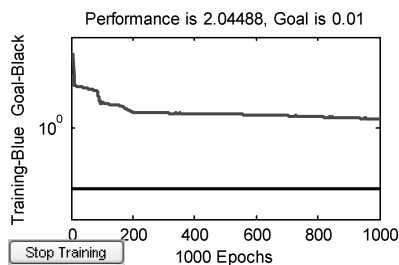


图 25-14 训练结果

网络训练结果为:

```
TRAINIDX, Epoch 0/1000, SSE 260.584/0.01, Gradient 984.285/1e-006
TRAINIDX, Epoch 20/1000, SSE 23.7826/0.01, Gradient 17.9028/1e-006
.....
TRAINIDX, Epoch 1000/1000, SSE 2.04488/0.01, Gradient 0.573174/1e-006
TRAINIDX, Maximum epoch reached, performance goal was not met.
```

4) 对网络进行测试

利用样本输入数据对训练好的网络进行测试:

```
A=sim(net,Pseq);
```

并绘出样本输入、目标输出和网络仿真输出曲线:

```
time=1:length(P);
plot(time,P,'.',time,T,'--',time,cat(2,A{:}));
title('Testing Amplitude Detection')
```

测试结果如图 25-13 所示,图中的虚线、短画线和实线分别表示网络输入的测试信号曲线、目标输出曲线和网络仿真输出曲线。从图 25-13 中可以看出网络对样本信号有较好的检测性能。

5) 对网络推广性的测试

利用一组新的输入数据对训练好的网络进行测试。首先产生输入信号和目标输出:



```
P3=sin(1:20)*2.6;
T3=ones(1,20)*2.6;
P4=sin(1:20)*1.2;
T4=ones(1,20)*1.2;
Pg=[P3 P4 P3 P4];
Tg=[T3 T4 T3 T4];
Pgseq=con2seq(Pg);
```

然后利用这一输入信号对网络进行仿真：

```
A=sim(net,Pgseq);
```

并绘出测试信号、目标输出和网络仿真输出曲线：

```
figure;
time=1:length(Pg);
plot(time,Pg,'-',time,Tg,'-',time,cat(2,A{:}));
title('Testing Generalization')
```

网络推广性能测试结果如图 25-15 所示，图中的虚线、短画线和实线分别表示网络输入的测试信号曲线、目标输出曲线和网络仿真输出曲线。从图 25-15 中可以看出网络对这一测试信号的检测性能虽然不及样本信号，但依然可以较成功地实现幅度检测。

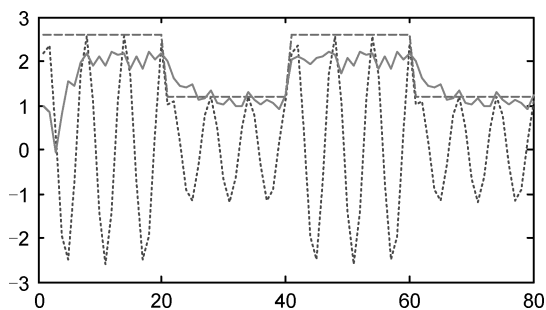


图 25-15 Elman 网络的推广性能

6) 完整的 MATLAB 代码

```
%产生样本数据 P 和目标输出 T
P1=sin(1:20);
P2=sin(1:20)*2;
T1=ones(1,20);
T2=ones(1,20)*2;
%将上述两组数据合并得到网络的输入和目标输出
P=[P1 P2 P1 P2];
T=[T1 T2 T1 T2];
Pseq=con2seq(P);
Tseq=con2seq(T);
%建立 Elman 网络，网络由两层神经元构成
```

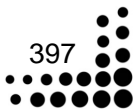


```
%两层分别有 10 个和 1 个神经元，传递函数分别为 tansig 函数和纯线性函数
%训练函数为 traingdx 函数
net=newelm([-2 2],[10 1],{'tansig','purelin'},'traingdx')
%对网络进行训练
net.trainParam.epochs=1000;
net.trainParam.show=20;
net.trainParam.goal=0.01;
net.performFcn='sse';
[net,tr]=train(net,Pseq,Tseq);
%利用样本数据对网络进行仿真
A=sim(net,Pseq);
time=1:length(P);
%画出样本数据的网络仿真输出图形
plot(time,P,'.',time,T,'--',time,cat(2,A{:}));
title('Testing Amplitude Detection')
%利用一组新的数据对网络进行检测
P3=sin(1:20)*2.6;
T3=ones(1,20)*2.6;
P4=sin(1:20)*1.2;
T4=ones(1,20)*1.2;
%产生测试数据
Pg=[P3 P4 P3 P4];
Tg=[T3 T4 T3 T4];
Pgseq=con2seq(Pg);
%利用测试数据对网络进行仿真
A=sim(net,Pgseq);
%画出测试数据的网络仿真输出图形
figure;
time=1:length(Pg);
plot(time,Pg,'.',time,Tg,'--',time,cat(2,A{:}));
title('Testing Generalization')
```

25.5 神经网络在噪声抵消系统中的应用

25.5.1 自适应噪声抵消原理

噪声消除是信号处理的核心问题之一，通常实现最优滤波的滤波器为维纳滤波器与卡尔曼滤波器。它们均要求已知信号和噪声的先验知识，但在许多实际应用中往往无法预先得知，为此发展了自适应滤波器。1965 年美国斯坦福大学建成了第一个自适应噪声抵消（ANC）系





统。随着计算机技术与集成电路技术的进步,新的自适应算法不断涌现出来,自适应噪声抵消在理论和应用上都得到了很大发展。自适应噪声抵消技术是一种能够很好地消除背景噪声影响的信号处理技术。应用自适应噪声抵消技术,可在未知外界干扰源特征、传递途径不断变化,以及背景噪声和被测对象声波相似的情况下,有效地消除外界声源的干扰,获得高信噪比的对象信号。这一技术可为机械元件的噪声、振动等动态信号在测试环境不太理想的工作现场做测试分析和故障诊断时提供有效的方法和依据,具有一定的理论意义和应用价值。

1) 自适应滤波器

自适应滤波器自从 20 世纪 60 年代出现后,其理论在不断发展与完善,应用也越来越广泛。自适应数字滤波器的原理如图 25-16 所示。图中:

$x(j)$ 表示 j 时刻的输入信号值;

$y(j)$ 表示 j 时刻的输出信号值;

$d(j)$ 表示 j 时刻的参考信号值或期望响应的信号值;

$e(j)$ 误差信号, $e(j) = d(j) - y(j)$ 。

自适应数字滤波器的滤波参数受误差信号 $e(j)$ 的控制,根据 $e(j)$ 的值自动调整,以便使输出 $y(j+1)$ 接近于所期望的参考信号 $d(j+1)$ 。

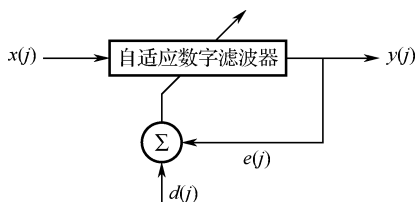


图 25-16 自适应数字滤波器原理

2) 自适应噪声抵消系统基本原理

自适应噪声抵消系统除了需要原始输入外,还需要一个参考输入,供给与原始输入相关的噪声,以便抵消原始输入中的噪声,而其中的有用信号几乎不产生什么畸变。

图 25-17 描述的是一个典型的自适应噪声抵消系统,其中原始输入信号 $d(k)$ 是有用信号 $s(k)$ 与噪声 $z(k)$ 之和,参考输入信号 $x(k)$ 是与 $z(k)$ 相关的噪声 $c(k)$ 。假设 $s(k)$ 、 $z(k)$ 与 $c(k)$ 是零均值的平稳随机过程, $s(k)$ 与 $z(k)$ 、 $c(k)$ 不相关。由图 25-17 可见,自适应滤波器的输出 $z_o(k)$ 为 $c(k)$ 的滤波信号,因此,自适应噪声抵消系统的输出 $y(k)$ 为:

$$y(k) = s(k) + z(k) - z_o(k)$$

而

$$y^2(k) = s^2(k) + [z(k) - z_o(k)]^2 + 2s(k)[z(k) - z_o(k)] \quad (25-1)$$

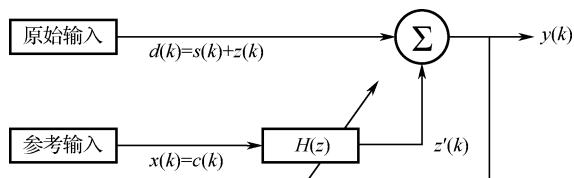


图 25-17 自适应噪声抵消系统



根据前面不相关的假定, 对式 (25-1) 两边取数学期望, 可得:

$$E[y^2(k)] = E[s^2(k)] + E[z(k) - zo(k)]^2$$

信号功率 $E[s^2(k)]$ 与自适应滤波器的调节无关, 因此, 自适应滤波器调节使 $E[y^2(k)]$ 最小, 也就是 $E[z(k) - zo(k)]^2$ 最小, $E[(y(k) - s^2(k))^2]$ 也最小, 即自适应噪声抵消系统的输出信号 $y(k)$ 与有用信号 $s(k)$ 的均方差最小。

在理想情况下, $zo(k) = z(k)$, 则 $y(k) = s(k)$, 这时, 自适应滤波器自动调节其脉冲响应, 将 $c(k)$ 加工成 $z(k)$, 与原始输入信号 $d(k)$ 中的 $z(k)$ 相减, 使输出信号 $y(k)$ 由于噪声完全被抵消, 而等于有用信号 $s(k)$ 。

可以证明, 自适应滤波器能完成上述任务的必要条件为: 参考输入信号 $x(k) = c(k)$ 必须与被抵消的信号 (现为噪声) $z(k)$ 相关。

一个实际的噪声抵消系统的情况比如图 25-17 所示的系统还要复杂一些, 这是因为输入还可能有一些独立的噪声源, 即与参考输入无关的噪声及干扰, 如图 25-18 所示。

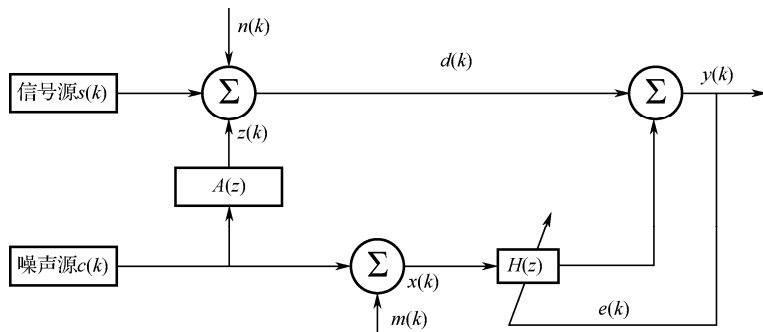


图 25-18 常见的自适应噪声抵消系统

25.5.2 噪声抵消系统的 MATLAB 仿真

1) BP 网络模型建立

根据分析结论, 这里构造一个 1-4-1 型 BP 神经网络, 隐含层节点取 4 个, 如图 25-19 所示。

本网络中的隐含层变换函数为 **tansig**, 是可微函数, 它可以将神经元的输入范围 $(-\infty, +\infty)$ 映射到 $(-1, +1)$, 非常适合于训练 BP 网络的神经元。如果 BP 网络的最后一层是 **Sigmoid** 型神经元, 那么整个网络的输出就会限制在一个较小的范围内; 如果是 **purelin** 型线性神经元, 则整个网络的输出可为任意值。

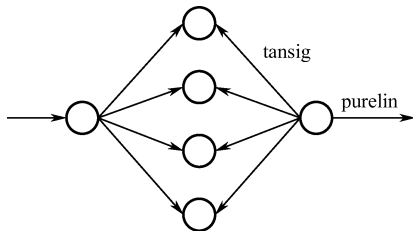


图 25-19 BP 神经网络模型

2) 基于神经网络工具箱的 BP 网络学习和训练

神经网络工具箱为训练神经网络提供了帮助, 可以利用它提供的函数对网络进行初始化、

仿真和训练，并通过变化的图形观察其动态训练过程。

图 25-20 给出了 BP 神经网络的训练过程。

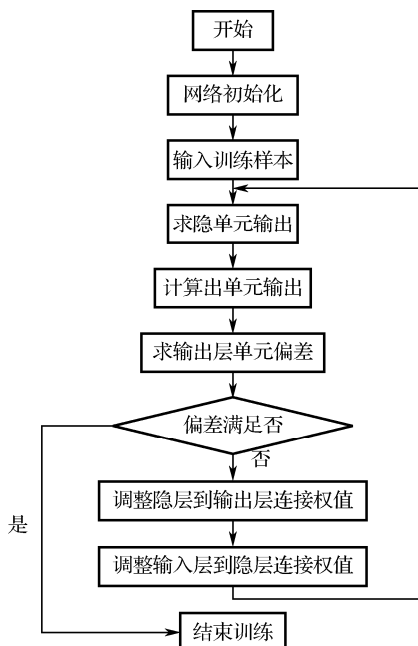


图 25-20 BP 神经网络训练过程

以下是 MATLAB 的实现代码：

```

%定义样本
T=0:0.01:10;
Y=randn(size(T));
Yn=sin(Y);
P=Y(1:30)
T=Yn(1:30)
%初始化 BP 网络
%其中，P 为输入量，隐层节点数为 4，输出层节点数为 1
%tansig 和 purelin 分别为隐层和输出层的变换函数
%训练算法为 trainlm
net=newff(minmax(P),[4,1],{'tansig','purelin'},'trainlm');
%BP 网络训练
%利用 BP 学习规则训练前向网络，使其完成函数逼近、向量分类和模式识别，选择训练参数，并
指示如何进行训练
%一旦训练达到最大的训练次数或网络误差平方和降低到误差之下，都会使网络停止学习
%指定两次更新显示间的训练次数
net.trainParam.show=10;
%指定训练的最大次数
net.trainParam.epochs=1000;
    
```




```
%误差平方和指标
net.trainParam.goal=0.001;
%指定学习速率，即权值和阈值更新的比例
net.trainParam.lr=0.01;
%开始训练
[net,tr]=train(net,P,T);
%画出误差变化曲线
figure(1);
plotperf(tr);
%计算网络仿真输出
A=sim(net,P);
figure(2);
plot([0:0.1:2.9],T,'b+-',[0:0.1:2.9],A,'r+*');
```

在这段程序中，BP 网络训练时，函数利用单层权值向量 W 、阈值量 B 及转移函数成批训练网络，使当输入 P 时，网络的输出为目标向量 T 。在这个环节将得到新的权值向量 W 和阈值 B ，以及记录网络训练过程的误差平方和行向量 tr 。

误差训练结果为：

```
TRAINLM, Epoch 0/1000, MSE 4.63725/0.001, Gradient 117.172/1e-010
TRAINLM, Epoch 3/1000, MSE 3.87303e-005/0.001, Gradient 0.0401599/1e-010
TRAINLM, Performance goal met.
```

误差变化曲线如图 25-21 所示，函数逼近曲线如图 25-22 所示。

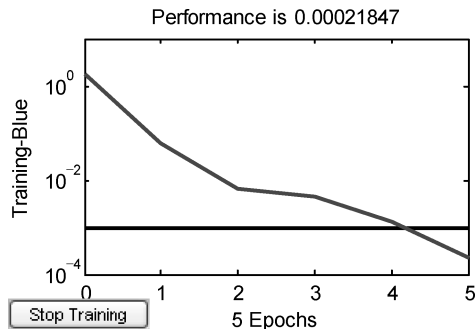


图 25-21 误差变化曲线

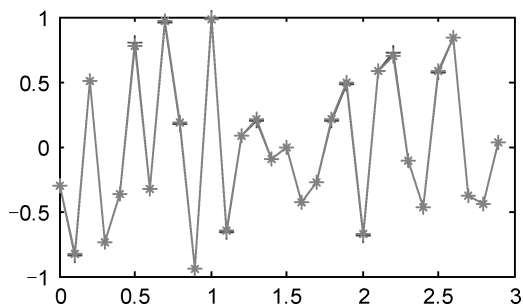


图 25-22 函数逼近曲线

根据误差曲线的变化，可以看到经过 600 多次训练之后，误差最终逼近于 0.01。从函数逼近曲线可看出，输出曲线非常逼近于目标曲线，证明训练后的网络具有较好的拟合性，在本环节中训练了网络并得到了构造 BP 网络的基本要素权值和阈值。

第 26 章 神经网络算法分析与工具箱应用

为了便于读者理解和应用 MATLAB 中的神经网络工具箱函数，首先介绍 MATLAB 中定义的神经网络对象及其属性。

在 MATLAB 中把定义的神经网络看作一个对象，对象还包括一些子对象：输入向量、网络层、输出向量、目标向量、权值向量和阈值向量等，这样网络对象和各子对象的属性共同确定了神经网络对象的特性。网络属性除了只读属性外，均可以按照约定的格式和属性值的类型进行设置、修改、引用等，只读属性只能引用。引用格式为：

网络名.[子对象].属性

例如，`net.numInputs`, `net.biasConnect(1)`, `net.inputConnect(1,2)`, `net.inputs{1}.range`。

为了说明网络对象、子对象及其属性，首先建立如图 26-1 和图 26-2 所示的神经网络 `net1` 和 `net2`。

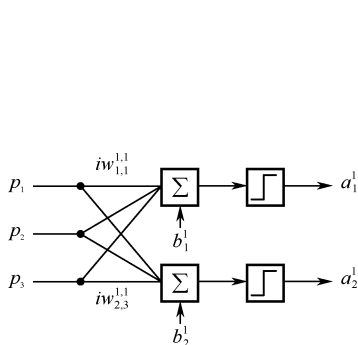


图 26-1 net1 的网络模型

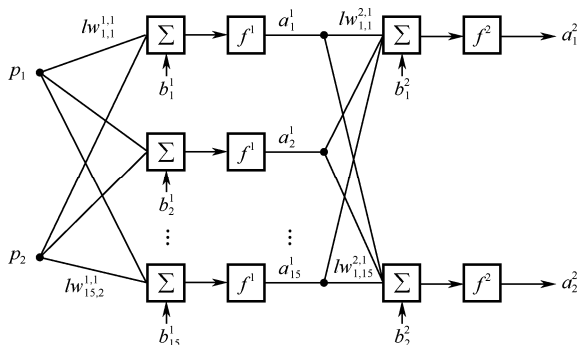


图 26-2 net2 的网络模型

在 MATLAB 命令窗口中逐条执行以下语句，即可创建网络 `net1` 和 `net2`。

```
>> p=[1 2;-1 1;0 1]
>> net1=newp(p,2)
net1 =
Neural Network object:
architecture:
    numInputs: 1
    numLayers: 1
    biasConnect: [1]
    inputConnect: [1]
    layerConnect: [0]
    outputConnect: [1]
```



```

targetConnect: [1]
    numOutputs: 1 (read-only)
    numTargets: 1 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)
subobject structures:
    inputs: {1x1 cell} of inputs
    layers: {1x1 cell} of layers
    outputs: {1x1 cell} containing 1 output
    targets: {1x1 cell} containing 1 target
    biases: {1x1 cell} containing 1 bias
    inputWeights: {1x1 cell} containing 1 input weight
    layerWeights: {1x1 cell} containing no layer weights
functions:
    adaptFcn: 'train'
    initFcn: 'initlay'
    performFcn: 'mae'
    trainFcn: 'trainc'
parameters:
    adaptParam: .passes
    initParam: (none)
    performParam: (none)
    trainParam: .epochs, .goal, .show, .time
weight and bias values:
    IW: {1x1 cell} containing 1 input weight matrix
    LW: {1x1 cell} containing no layer weight matrices
    b: {1x1 cell} containing 1 bias vector
other:
    userdata: (user stuff)
>> net2=newff([-1 1;-1 1],[15 2],{'tansig' 'purelin'},'traingdx','learnqdm')
net2 =
Neural Network object:
architecture:
    numInputs: 1
    numLayers: 2
    biasConnect: [1; 1]
    inputConnect: [1; 0]
    layerConnect: [0 0; 1 0]
    outputConnect: [0 1]
    targetConnect: [0 1]
    numOutputs: 1 (read-only)
    numTargets: 1 (read-only)

```





```
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)
subobject structures:
    inputs: {1x1 cell} of inputs
    layers: {2x1 cell} of layers
    outputs: {1x2 cell} containing 1 output
    targets: {1x2 cell} containing 1 target
    biases: {2x1 cell} containing 2 biases
    inputWeights: {2x1 cell} containing 1 input weight
    layerWeights: {2x2 cell} containing 1 layer weight
functions:
    adaptFcn: 'trains'
    initFcn: 'initlay'
    performFcn: 'mse'
    trainFcn: 'traingdx'
parameters:
    adaptParam: .passes
    initParam: (none)
    performParam: (none)
    trainParam: .epochs, .goal, .lr, .lr_dec,
                .lr_inc, .max_fail, .max_perf_inc, .mc,
                .min_grad, .show, .time
weight and bias values:
    IW: {2x1 cell} containing 1 input weight matrix
    LW: {2x2 cell} containing 1 layer weight matrix
    b: {2x1 cell} containing 2 bias vectors
other:
    userdata: (user stuff)
```

26.1 网络对象属性

从上面创建函数 `newp` 和 `newff` 所创建的 `net1` 和 `net2` 后显示结果可看出所定义的神经网络 `net1` 和 `net2` 的属性。

26.1.1 结构属性

结构属性决定了网络子对象的数目（包括输入向量、网络层向量、输出向量、目标向量、权值向量和阈值向量的数目）以及它们的连接关系。无论何时，结构属性值一旦发生变化，网络就会自动重新定义，与之相关的其他属性值也会自动更新。



1) numInputs 属性

`net.numInputs` 属性定义了网络的输入向量数，它可以被设置为 0 或正整数，其值一般在用户自定义网络中才被设置，而由 MATLAB 神经网络工具箱中的网络定义函数创建的网络往往是单个网络，其输入向量只有一个。如果用户定义了一个由多个网络构成的复杂的网络，其输入向量就不止一个，而是多个；而每个输入向量也可能包含了多个元素，其元素的个数由输入向量的大小 (`net.inputs{i}, size`) 决定，所以网络的输入向量数并不是网络输入元素（变量）的个数，这一点希望引起读者的注意。

2) numLayers 属性

`net.numLayers` 属性定义了网络的层数，它可以被设置为 0 或正整数。

例如，上节显示结果中，`net1.numLayers=1` 表示 `net1` 只有一个网络层；`net2.numLayers=2` 表示 `net2` 有两个网络层。

`net.numLayers` 属性值一旦改变，下列与网络层相关的布尔代数矩阵的大小都会随之改变：

```
net.biasConnect
net.inputConnect
net.layerConnect
net.outputConnect
net.targetConnect
```

下列与网络层相关的子对象细胞矩阵的大小也会随之改变：

```
net.biases
net.inputWeights
net.layerWeights
net.outputs
net.targets
```

下列网络调整参数细胞矩阵的大小也会随之改变：

```
net.IW
net.LW
net.b
```

上面提到的细胞矩阵的概念可能对有的读者比较陌生，其实很简单，它将多个矩阵向量作为细胞矩阵的一个“细胞”，细胞矩阵的各个元素值为对应细胞的大小和数值类型，为区别于其他矩阵，用 {} 表示细胞矩阵。例如，设矩阵：

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

若将 a, b, c 分别作为细胞矩阵的一个细胞，则可以定义细胞矩阵 m, n 为：

$$m = \begin{Bmatrix} a \\ b \end{Bmatrix} = \begin{Bmatrix} 2 \times 2 & \text{double} \\ 2 \times 3 & \text{double} \end{Bmatrix}, n = \begin{Bmatrix} a & [] \\ b & c \end{Bmatrix} = \begin{Bmatrix} [2 \times 2 \text{double}] & [] \\ [2 \times 3 \text{double}] & [3 \times 2 \text{double}] \end{Bmatrix}$$



可以通过下标，访问细胞矩阵中各细胞的值：

$$m\{1\} = n\{1,1\} = a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, m\{2\} = n\{2,1\} = b = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$n\{2,2\} = c = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, n\{1,2\} = []$$

3) biasConnect 属性

`net.biasConnect` 属性定义各个网络层是否具有阈值向量，其值为 $N_l \times 1$ 布尔型向量（0 或 1）， N_l 为网络层数（`net.numLayers`）。

可以通过访问 `net.biasConnect{i}` 的值，查看第 i 个网络层是否具有阈值向量。

`net.baisConnect` 属性值一旦改变，则阈值结构细胞矩阵（`net.biases`）和阈值向量细胞矩阵（`net.b`）将随之改变。

4) inputConnect 属性

`net.inputConnect` 属性定义各网络层是否具有来自各输入向量的连接权，其值为 $N_l \times N_i$ 的布尔型向量（0 或 1）， N_l 为网络层数（`net.numLayers`）， N_i 为网络输入向量数（`net.numLaypers`）。

可以通过访问 `net.inputConnect(i,j)` 的值，查看第 i 个网络层是否具有来自第 j 个输入向量的连接权。

`net.inputConnect` 属性值一旦改变，输入层权值结构细胞矩阵（`net.inputWeights`）和权值向量细胞矩阵（`net.IW`）将随之改变。

5) layerConnect 属性

`net.layerConnect` 属性定义一个网络层是否具有来自另外一个网络层的连接权，其值为 $N_l \times N_l$ 的布尔型向量（0 或 1）， N_l 为网络层数（`net.numLaypers`）。

可以通过访问 `net.layerConnect(i,j)` 的值，查看第 i 个网络层是否具有来自第 j 个网络层的连接权。例如，在前面的显示结果中，`net1.layperConnect=[0]` 和 `net2.layerConnect=[0 0; 1 0]`，说明 `net1` 唯一的一个网络层没有来自本层的连接权（无层内连接）；`net2` 的第二个网络层具有来自第一个网络层的连接权，而第一个网络层没有来自第二个网络层的连接权（无反馈），各网络层也没有来自本层的连接权（无层内连接）。

`net.layerConnect` 属性值一旦改变，网络层权值结构细胞矩阵（`net.layerWeights`）和网络层权值向量细胞矩阵（`net.IW`）将随之改变。

6) outputConnect 属性

`net.outputConnect` 属性定义各网络层是否作为输出层，其值为 $1 \times N_l$ 的布尔型向量（0 或 1）， N_l 为网络层数（`net.numLayers`）。

可以通过访问 `net.outputConnect(i)` 的值，查看第 i 个网络层是否作为输出层。例如，在前面的显示结果中，`net1.outputConnect=[1]` 和 `net2.outputConnect=[0 1]` 说明 `net1` 唯一的一个网络层也为输出层，`net2` 的第二个网络层为输出层，而第一个网络层不是输出层。



`net.outputConnect` 属性值一旦改变, 网络层输出层的数目 (`net.numOutputs`) 和输出层结构细胞矩阵 (`net.outputs`) 将随之改变。

7) `targetConnect` 属性

`net.targetConnect` 属性定义各网络层是否和目标向量有关, 其值为 $1 \times N_l$ 的布尔型向量 (0 或 1), N_l 为网络层数 (`net.numLayers`)。

可以通过访问 `net.targetConnect(i)` 的值, 查看第 i 个网络层是否和目标向量有关。例如在前面的显结果中, `net1.targetConnect=[1]` 和 `net1.outputConnect=[0 1]` 说明 `net1` 唯一的一个网络层和目标向量有关; `net2` 的第二个网络层和目标向量有关, 而第一个网络层和目标向量没有关系。

8) `numTargets` 属性 (只读)

`net.numTargets` 属性值为目标向量的数目, 它等于 `targetConnect` 矩阵中元素值为 (`True`) 的个数之和, 即

```
numTargets=sum(net.targetConnect)
```

9) `numOutputs` 属性 (只读)

`net.numOutputs`, 该属性值为输出向量的数目, 它等于 `outputConnect` 矩阵中元素值为 1 (`True`) 的个数之和, 即

```
numOutputs=sum(net.outputConnect)
```

10) `numInputDelays` 属性 (只读)

`net.numInputDelays` 属性定义进行网络仿真时输入向量的延迟量, 其值总是设置为与网络输入相连接的权值延迟量的最大值, 即

```
numInputDelays=0;
for i=1:net.numLayers
    for j=1:net.numInputs
        if net.inputConnect(i,j)
            numInputDelays=max(...
                [numInputDelays net.inputWeights{i,j}.delays]);
        end
    end
end
```

例如, 在前面的显示结果中, `net1.numTargets` 和 `net2.numTargets` 均为 0, 说明输入向量无延迟。

11) `numLayerDelays` 属性 (只读)

`net.numLayerDelays` 属性定义进行网络仿真时网络层输出单元的延迟量, 其值总是设置为与网络层相连接的权值延迟量的最大值, 即



```

numLayerDelays=0;
for i=1:net.numLayers
    for j=1:net.numLayers
        if net.layerConnect(i,j)
            numLayerDelays=max(...
                [numLayerDelays net.layerWeights{i,j}.delays]);
        end
    end
end
end

```

例如，在前面的显示结果中，`net1.numTargets` 和 `net2.numTargets` 均为 0，说明网络层输出单元无延迟。

26.1.2 子对象结构属性

子对象结构属性定义了下列细胞矩阵：每个网络的输入向量结构、网络层结构、输出向量结构、目标向量结构、阈值向量结构和权值向量结构细胞矩阵。

1) inputs 属性（输入层结构）

`net.inputs` 属性定义网络输入向量的结构，其值为关于网络各输入向量结构的细胞矩阵，大小为 $N_i \times 1$ ， N_i 为输入向量数（`net.numInputs`）。

例如，在前面的显示结果中，`net1.inputs` 和 `net2.inputs` 均为 $\{1 \times 1 \text{ cell}\}$ ，说明网络输入向量的结构是 1×1 的细胞矩阵，即只有一个输入向量：

```
net1.inputs=net2.inputs={ [1x1 struct] }
```

每个输入向量的属性结构，可以通过 `net.inputs{i}` 进行访问。例如在前面的显示结果中：

$$\begin{aligned} \text{net1.inputs}\{1\} &= \begin{bmatrix} \text{range}:[3 \times 2\text{double}] \\ \text{size}:3 \\ \text{userdata}:[1 \times 1\text{struct}] \end{bmatrix} \\ \text{net2.inputs}\{1\} &= \begin{bmatrix} \text{range}:[2 \times 2\text{double}] \\ \text{size}:2 \\ \text{userdata}:[1 \times 1\text{struct}] \end{bmatrix} \end{aligned}$$

每个输入向量的属性值可以通过 `net.inputs{i}.属性名` 进行访问。例如：

$$\text{net1.inputs}\{1\}.\text{range} = \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 0 & 1 \end{bmatrix}$$

2) layers 属性

`net.layers` 属性定义各网络层的结构特性，其值为关于网络层结构的细胞矩阵，大小为 $N_l \times 1$ ， N_l 为网络层数（`net.numLayers`）。



例如,在前面的显示结果中,net1.layers 为{1×1 cell}说明网络层结构是 1×1 的细胞矩阵,即只有一个网络层;net2.layers 为{2×1 cell}说明网络层结构是 2×1 的细胞矩阵,即有两个网络层:

```
net1.layers = {[1×1 struct]}
```

```
net2.layers = [ [1×1 struct]
                [1×1 struct] ]
```

每个网络层的结构属性可以通过 net.layers{i} 访问。例如:

```
net1.layers{1} =
    dimensions:2
    distanceFcn:
    distances:[]
    initFcn:'initwb'
    netInputFcn:'netsum'
    positions:[0 1]
    size:2
    topologyFcn:'hextop'
    transferFcn:'hardlim'
    userdata:[1×1 struct]
```

每个网络层的结构属性值,可以通过 net.layers{i}.属性名访问。例如:

```
net1.layers{1}.transferFcn=hardlim
```

3) outputs 属性

net.outputs 属性定义各网络层作为输出向量的结构特性,其值为关于输出向量结构的细胞矩阵,大小为 $N_l \times 1$, N_l 为网络层数 (net.numLayers)。

每个输出向量的结构属性可以通过 net.outputs{i} 访问。例如,

```
net1.outputs{1} =
    size:2
    userdata:[1×1 struct]
```

每个输出向量的结构属性值,可以通过 net.outputs{i}.属性名访问。例如:

```
net1.outputs{1}.size=2
```

4) targets 属性

net.targets 属性定义各网络层目标向量的结构特性,其值为关于目标向量结构的细胞矩阵,大小为 $N_l \times 1$, N_l 为网络层数 (net.numLayers)。

每个网络层目标向量的结构属性可以通过 net.targets{i} 访问。例如,

```
net2.targets{2} =
    size:2
    userdata:[1×1 struct]
```

每个目标向量的结构属性值可以通过 net.targets{i}.属性名访问。例如:

```
net2.targets{2}.size=2
```



5) biases 属性

`net.biases` 属性定义各网络层阈值向量的结构特性, 其值为关于阈值向量结构的细胞矩阵, 大小为 $N_l \times 1$, N_l 为网络层数 (`net.numLayers`)。

例如, 在前面的显示结果中, `net1.biases` 为 $\{1 \times 1 \text{ cell}\}$ containing 1 bias 说明阈值向量结构是 1×1 的细胞矩阵, 只有一个阈值向量; `net2.layers` 为 $\{2 \times 1 \text{ cell}\}$ containing 2 biases 说明阈值向量结构是 2×1 的细胞矩阵, 即有两个网络层都具有阈值向量:

$$\text{net1.biases} = \{[1 \times 1 \text{ struct}]\}$$

$$\text{net2.biases} = \begin{Bmatrix} [1 \times 1 \text{ struct}] \\ [1 \times 1 \text{ struct}] \end{Bmatrix}$$

每个网络层阈值向量的结构属性可以通过 `net.biases{i}` 访问。例如:

$$\text{net2.biases}\{2\} = \begin{bmatrix} \text{initFcn} : '' \\ \text{learn} : 1 \\ \text{learnFcn} : 'learn\text{gdm}' \\ \text{learnParam} : [1 \times 1 \text{ struct}] \\ \text{size} : 2 \\ \text{userdata} : [1 \times 1 \text{ struct}] \end{bmatrix}$$

每个阈值向量的结构属性值, 可以通过 `net.biases{i}.属性名` 访问。例如:

```
net2.biases{2}.learnFcn=learn\text{gdm}
```

6) inputWeights 属性

`net.inputWeights` 属性定义输入层权值向量的结构特性, 其值为关于输入层权值向量结构的细胞矩阵, 大小为 $N_l \times N_i$, N_l 为网络层数 (`net.numLayers`); N_i 为输入向量数 (`net.numInputs`)。

例如, 在前面的显示结果中, `net1.inputWeights` 为 $\{1 \times 1 \text{ cell}\}$ containing 1 input weight, 说明输入层的权值向量结构是 1×1 的细胞矩阵, 只有一个输入层权值向量; `net2.layers` 为 $\{2 \times 1 \text{ cell}\}$ containing 1 input weights, 说明阈值向量结构是 2×1 的细胞矩阵, 即有两个网络层, 但只有一个输入层权值向量:

$$\text{net1.inputWeights} = \{[1 \times 1 \text{ struct}]\}$$

$$\text{net2.inputWeights} = [1 \times 1 \text{ struct}]$$

可以通过访问 `net.inputWeights{i,j}`, 了解从第 j 个输入向量连接到第 i 个网络层的结构属性。例如:



$$\text{net2.inputWeights}\{1,1\} = \begin{bmatrix} \text{delays} : 0 \\ \text{initFcn} : '' \\ \text{learn} : 1 \\ \text{learnFcn} : \text{'leangdm'} \\ \text{learnParam} : [1 \times 1 \text{ struct}] \\ \text{size} : [15 \ 2] \\ \text{userdata} : [1 \times 1 \text{ struct}] \\ \text{weightFcn} : \text{'dotprod'} \end{bmatrix}$$

每个输入层权值向量的结构属性值，可以通过 `net.inputWeights{i,j}` 属性名访问，例如：

```
net2.inputWeights{1,1}.learnFcn=leangdm
```

7) layerWeights 属性

`net.layerWeights` 属性定义各网络层权值向量的结构特性，其值为关于各网络层权值向量结构的细胞矩阵，大小为 $N_l \times N_l$ ， N_l 为网络层数（`net.numLayers`）。

例如，在前面的显示结果中，`net1.layerWeights` 为 $\{1 \times 1 \text{ cell}\}$ containing no layer weights，说明网络层的权值向量结构是 1×1 的细胞矩阵，但没有网络层权值向量；`net2.layerWeights` 为 $\{2 \times 2 \text{ cell}\}$ containing 1 layer weight，说明网络层权值向量结构是 2×2 的细胞矩阵，即有两个网络层，但只有一个网络层权值向量：

$$\begin{aligned} \text{net1.layerWeights} &= \{\{\}\} \\ \text{net2.layerWeights} &= \begin{bmatrix} \square & \square \\ [1 \times 1 \text{ struct}] & \square \end{bmatrix} \end{aligned}$$

`net2.layerWeight{2,1}` 不为空值，说明有第 1 网络层到第 2 网络层的权值向量。网络层权值向量的结构属性可以由 `net.layerWeights{i,j}` 查看。例如：

$$\text{net2.layerWeights}\{2,1\} = \begin{bmatrix} \text{delays} : 0 \\ \text{initFcn} : '' \\ \text{learn} : 1 \\ \text{learnFcn} : \text{'leangdm'} \\ \text{learnParam} : [1 \times 1 \text{ struct}] \\ \text{size} : [2 \ 15] \\ \text{userdata} : [1 \times 1 \text{ struct}] \\ \text{weightFcn} : \text{'dotprod'} \end{bmatrix}$$

每个网络层权值向量的结构属性值，可以通过 `net.layerWeights{i,j}` 属性名访问。例如：

```
net2.layerWeights{2,1}.learnFcn=leangdm
```

26.1.3 函数属性

函数属性定义了一个网络在进行权值/阈值调整、初始化、误差性能计算或训练时采用的



算法。

1) initFcn 属性

`net.initFcn` 属性定义了网络初始化权值/阈值向量时所采用的函数，它可以被设置为任意一个进行网络权值/阈值初始化的函数名，包括 `initlay`（网络层初始化函数）工具箱函数。

`init` 函数一旦被调用，就可以实现网络权值/阈值的初始化：

```
net=init(net)
```

另外，用户可以自定义权值/阈值初始化函数。

`init` 属性值一旦发生变化，网络的初始化参数（`net.initParam`）将被设置为新的初始化函数所包含的参数及其默认参数值。

2) adaptFcn 属性

`net.adaptFcn` 属性定义了网络进行权值/阈值调整所采用的函数，它可以被设置为任意一个进行权值/阈值调整的函数名，包括 `trains` 函数。

`adapt` 函数一旦被调用，就可以实现网络权值/阈值的调整：

```
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)
```

另外，用户可以自定义权值/阈值调整函数。

`adaptFcn` 属性值一旦发生变化，网络的调整参数（`net.adaptParam`）将被设置为新的调整函数所包含的参数及其默认参数值。

3) performFcn 属性

`net.performFcn` 属性定义了网络用于衡量网络性能所采用的函数，它可以被设置为任意一个网络性能函数名（见表 26-1）。

表 26-1 MATLAB 工具箱误差性能函数

函 数 名	说 明
mae	绝对平均误差性能函数
mse	均方差性能函数
msereg	归一化均方差性能函数
sse	平方和误差性能函数

当调用 `train` 函数时，上述性能函数被用于训练过程中的性能计算：

```
[net, tr]=train(NET, P, T, Pi, Ai)
```

另外，用户可以自定义性能函数。

`performFcn` 属性值一旦发生变化，网络性能参数（`net.performParam`）将被设置为新的性能函数所包含的参数及其默认参数值。



4) trainFcn 属性

net.trainFcn 属性定义了网络用于训练网络性能所采用的函数，它可以被设置为任意一个训练函数名（见表 26-2）。

表 26-2 MATLAB 工具箱的训练函数

函 数 名	说 明
trainbfg	BFGS 算法（拟牛顿反向传播算法）训练函数
trainbr	贝叶斯归一化法训练函数
traincgb	Powell-Beale 共轭梯度反向传播算法训练函数
traincgf	Fletcher-Powell 变梯度反向传播算法训练函数
traincgp	Polak-Ribiere 变梯度反向传播算法训练函数
traingd	梯度下降反向传播算法训练函数
traingda	自适应调整学习率的梯度下降反向传播算法训练函数
traingdm	附加动量因子的梯度下降反向传播算法训练函数
traingdx	自适应调整学习率并附加动量因子的梯度下降反向传播算法训练函数
trainlm	Levenberg-Marquardt 反向传播算法训练函数
trainoss	OSS（one-step secant）反向传播算法训练函数
trainrp	RPROP（弹性 BP 算法）反向传播算法训练函数
trainscg	SCG（scaled conjugate gradient）反向传播算法训练函数
trianb	以权值/阈值的学习规则采用批处理的方式进行训练的函数
trainc	以学习函数依次对输入样本进行训练的函数
trainr	以学习函数随机对输入样本进行训练的函数

当调用 train 函数时，上述训练函数被用于训练网络：

```
[net, tr]=train(NET, P, T, Pi, Ai)
```

另外，用户可以自定义训练函数。

trainFcn 属性值一旦发生变化，网络训练参数（net.trainParam）将被设置为新的训练函数所包含的参数及其默认参数值。

26.1.4 权值和阈值

权值和阈值属性定义了网络的可调整参数：权值向量和阈值向量。

1) IW 属性

net.IW 属性定义从网络输入向量到网络层的权值向量（输入层的权值向量）结构。其值为 $N_l \times N_i$ 的细胞矩阵， N_l 为网络层数（net.numLayers）， N_i 为输入向量数（net.numInputs）。例如，前面的显示结果中，net1.IW 为{1×1 cell} containing 1 input weight matrix，说明输入层



的权值向量结构是 1×1 的细胞矩阵，有一个输入层权值向量；net2.IW 为 $\{2 \times 1 \text{ cell}\}$ containing 1 input weight matrix，说明输入层的权值向量结构是 2×1 的细胞矩阵，有一个输入层权值向量。net.IW 的元素值说明了各输入层权值向量的结构和数值类型：

$$\begin{aligned} \text{net1.IW} &= \{[2 \times 3 \text{double}]\} \\ \text{net2.IW} &= \begin{Bmatrix} [15 \times 2 \text{double}] \\ [] \end{Bmatrix} \end{aligned}$$

通过访问 net.IW{i,j}，可以获得第 i 个网络层来自第 j 个输入向量的权值向量值。例如：

$$\text{net1.IW}\{1,1\} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

2) b 属性

net.b 属性定义各网络层的阈值向量结构，其值为 $N_l \times 1$ 的细胞矩阵， N_l 为网络层数 (net.numLayers)。例如，在前面的显示结果中，net1.b 为 $\{1 \times 1 \text{ cell}\}$ containing 1 bias vector，说明阈值向量结构是 1×1 的细胞矩阵，有一个阈值向量；net2.b 为 $\{2 \times 1 \text{ cell}\}$ containing 2 bias vectors，说明阈值向量结构是 2×2 的细胞矩阵，含有两个阈值向量。net.b 的元素值说明各网络层阈值向量的结构和数值类型：

$$\begin{aligned} \text{net1.b} &= \{[2 \times 1 \text{double}]\} \\ \text{net2.b} &= \begin{Bmatrix} [15 \times 1 \text{double}] \\ [2 \times 1 \text{double}] \end{Bmatrix} \end{aligned}$$

通过访问 net.b{i}，可以获得第 i 个网络层阈值向量。

3) LW 属性

net.LW 属性定义从一个网络层到另一个网络层的权值向量结构，其值为 $N_l \times N_l$ 的细胞矩阵， N_l 为网络层数。例如，在前面的显示结果中，net1.LW 为 $\{1 \times 1 \text{ cell}\}$ containing no layer weight matrices，说明网络层的权值向量结构是 1×1 的细胞矩阵，但没有网络层之间的权值向量；net2.IW 为 $\{2 \times 2 \text{ cell}\}$ containing 1 layer weight matrix，说明输入层的权值向量结构是 2×2 的细胞矩阵，含有一个网络层之间的连接权值向量。net1.LW 的元素值说明了各网络层权值向量的结构和数值类型：

$$\begin{aligned} \text{net1.LW} &= \{[]\} \\ \text{net2.LW} &= \begin{Bmatrix} [] & [] \\ [2 \times 15 \text{double}] & [] \end{Bmatrix} \end{aligned}$$

只有 net.LW{2,1} 不为空值，说明只有第 2 网络层来自第 1 网络层的权值向量。

通过访问 net.LW{i,j}，可以获得第 i 个网络层来自第 j 个网络层的权值向量值。



26.1.5 参数属性

1) adaptParam 属性

`net.adaptParam` 属性定义当前网络权值/阈值调整函数的参数及参数值，取决于当前的权值/阈值调整函数（`net.adaptFcn`），可以查看有关调整函数的帮助获得这些参数及参数值。在 MATLAB 的命令窗口输入命令：

```
help(net.adaptFcn)
```

也可以获得这些参数及参数值的具体描述。

2) initParam 属性

`net.initParam` 属性定义当前初始化函数参数及参数值，取决于当前的初始化函数（`net.initFcn`），可以查看有关初始化函数的帮助获得这些参数及参数值。在 MATLAB 命令窗口输入命令：

```
help(net.initFcn)
```

也可以获得这些参数及参数值的具体描述。

3) performParam 属性

`net.PerformParam` 属性定义当前性能函数的参数及参数值，取决于当前的性能函数（`net.performFcn`），可以查看有关性能函数的帮助获得这些参数及参数值。在 MATLAB 命令窗口中输入命令：

```
help(net.performFcn)
```

也可以获得这些参数及参数值的具体描述。

4) trainParam 属性

`net.trainParam` 属性定义当前训练函数的参数及参数值，取决于当前的训练函数（`net.trainFcn`），可以查看有关训练函数的帮助获得这些参数及参数值。在 MATLAB 命令窗口输入命令：

```
help(net.trainFcn)
```

也可以获得这些参数及参数值的具体描述。

26.1.6 其他属性

`net.userdata` 属性为用户提供了增加关于网络对象用户信息的地方，它预先只定义了一个字段，其值为一段提示信息：

```
net.userdata=note:'Put your custom network information here'
```



用户可以通过修改 `net.userdata.note` 的值，增加关于网络对象的用户信息。

26.2 子对象属性

26.2.1 输入向量

`net.inputs`，该子对象的属性详细定义了网络的每个输入向量的每一个输入量。例如：

```
net1.inputs=net2.inputs=[1×1 struct]
```

通过访问 `net.inputs{i}` 可以获得第 i 个输入向量的属性值：

$$\begin{aligned} \text{net1.inputs}\{1\} &= \begin{bmatrix} \text{range}:[3 \times 2 \text{double}] \\ \text{size}:3 \\ \text{userdata}:[1 \times 1 \text{struct}] \end{bmatrix} \\ \text{net2.inputs}\{1\} &= \begin{bmatrix} \text{range}:[2 \times 2 \text{double}] \\ \text{size}:2 \\ \text{userdata}:[1 \times 1 \text{struct}] \end{bmatrix} \end{aligned}$$

1) range 属性

`net.inputs{i}.range` 定义了第 i 个输入向量中每个元素的取值范围，其值是一个 $R \times 2$ 的矩阵， R 为输入向量的元素个数。矩阵的第一列为每个元素的最小值，第二列为每个元素的最大值。它包含两个信息：①输入向量 R 的元素个数（亦即输入变量的个数）；②每个输入变量的取值区间。从而确定了输入向量的规模，另外 `range` 属性值还用于在一些初始化函数中确定连接输入向量的权值和阈值的初值。

当 `net.inputs{i}.range` 的行数变化时，表明输入向量的元素数目发生了改变，那么网络输入向量的 `size` 属性值（`net.inputs{i}.size`）、与之相连接的权值的 `size` 属性值（`net.inputWeights(:,i).size`）及输入权值向量（`net.IW(:,i)`）的大小会自动做相应的变化。

2) size 属性

`net.inputs{i}.size` 定义了网络各输入向量的元素数目，可被设置为 0 或正整数。当其值发生变化时，表明输入向量的元素数目发生了变化，那么相应的 `range` 属性值（`net.inpust{i}.range`）、与之相连接的权值的 `size` 属性值（`net.inputWeights(:,i).size`）及输入权值向量（`net.IW(:,i).size`）的大小会自动做相应的变化。

3) userdata 属性

`net.inputs{i}.userdata` 和 `net.userdata` 为用户提供了增加关于输入向量用户信息的地方，它预先只定义了一个字段，其值为一段提示信息。

```
net.inputs{i}.userdata=note:'Put your custom input information here.'
```




用户可以通过修改 `net.inputs{i}.userdata.note` 的值增加关于输入向量的用户信息。

26.2.2 网络层

`net.layers`，该子对象的属性详细定义了网络的每一个网络层。例如：

```
net1.layers = {[1×1struct]}
```

```
net2.layers = {[1×1struct]
               [1×1struct]}
```

通过访问 `net.layers{i}` 可以获得第 i 个网络层的属性值：

```
net1.layers{1} =
    dimensions : 2
    distanceFcn : "
    distances : []
    initFcn : 'initwb'
    netInputFcn : 'netsum'
    position : [0 1]
    size : 2
    topologyFcn : 'hextop'
    transferFcn : 'hardlim'
    userdata : [1×1struct]

net2.layers{2} =
    dimensions : 2
    distanceFcn : "
    distances : []
    initFcn : 'initwb'
    netInputFcn : 'netsum'
    position : [0 1]
    size : 2
    topologyFcn : 'hextop'
    transferFcn : 'purelin'
    userdata : [1×1struct]
```

1) dimensions 属性

`net.layers{i}.dimensions` 属性定义了第 i 个网络层神经元的维数。对于自组织映射的多维方式，能够设置网络层的神经元维数是很重要的。`net.layers{i}.dimensions` 可以被设置成所有元素的值为 0 或正整数的行向量，此时，行向量所有元素的乘积即该网络层的神经元数。

当采用网络层拓扑函数（`net.layers{i}.topologyFcn`）计算神经元在网络层中的位置（`net.layers{i}.positions`）时，将用到网络层神经元维数。

`net.layers{i}.dimensions` 属性一旦改变，网络层的大小（`net.layers{i}.size`）、网络层神经元的位置（`net.layers{i}.positions`）以及两个神经元之间的距离（`net.layers{i}.distances`）都会随之





改变。

2) distanceFcn 属性

`net.layers{i}.distanceFcn`，该属性定义一函数，用于第 i 个网络层中的神经元之间距离的计算，它是根据神经元的位置 (`net.layers{i}.position`) 来进行计算的。神经元的距离用于自组织映射神经网络。

该属性可被设置成神经网络工具箱中的任意一个距离函数名 (见表 26-3)。

表 26-3 MATLAB 神经网络工具箱的距离计算函数

函 数	说 明
<code>boxdist</code>	计算两个位置向量之间的距离函数
<code>dist</code>	欧几里得 (Euclidean) 距离权值函数
<code>linkdist</code>	连接距离函数
<code>mandist</code>	曼哈顿 (Manhattan) 距离权值函数

除了以上距离函数外，用户还可以自定义距离函数，相关内容读者可以参考相关文献。

`net.layers{i}.distanceFcn` 属性一旦发生改变，网络层神经元之间的距离 (`net.layers{i}.distances`) 将随之改变。

3) initFcn 属性

`net.layers{i}.initFcn`，如果网络初始化函数 (`net.initFcn`) 设置为 `initlay`，则该属性定义第 i 个网络层的初始化函数。

该属性可被设置为任意一个神经网络工具箱中的网络层初始化函数名 (见表 26-4)。

表 26-4 MATLAB 神经网络工具箱的网络层初始化函数

函 数	说 明
<code>initnw</code>	NW (Nguyen-Window) 网络层初始化函数
<code>initwb</code>	通过权值和阈值进行网络层初始化函数

如果网络初始化函数设置为 `initlay`，那么，当 `init` 函数被调用时，该属性定义的网络层初始化函数将用于对网络层的权值和阈值的初始化：

```
net=init(net)
```

除了以上网络层初始化函数，用户可以自定义网络层初始化函数。

4) distances 属性 (只读)

`net.layers{i}.distances`，该属性定义第 i 个网络层中神经元之间的距离，这些距离用于自组织映射神经网络。其值为网络层距离函数 (`net.layers{i}.distanceFcn`) 的计算结果，是通过网络层神经元的位置 (`net.layers{i}.position`) 进行计算的。



5) positions 属性 (只读)

`net.layers{i}.positions`, 该属性定义第 i 个网络层中神经元的位置, 这些位置用于自组织映射。其值为网络层拓扑结构函数 (`net.layers{i}.topologFcn`) 关于位置的计算结果, 它是通过网络层神经元维数 (`net.layers{i}.dimensions`) 进行计算的。

可以用 `plotsom` 函数画出网络层神经元的位置图。例如:

```
plotsom(net2.layers{1}.positions)
```

`net2` 第 1 网络层的位置如图 26-3 所示。`net2.layers{1}.dimensions=[15]` 是一维的, 所以所有神经元的位置都在水平轴上, 即 `positions(2,i)=0`。

其总的程序代码如下:

```
p=[1 2;-1 1;0 1];
net1=newp(p,2);
net2=newff([-1 1;-1 1],[15 2],{'tansig' 'purelin'},'traingdx','learnqdm');
net2.layers{1}.dimensions=[15];
for i=1:15
    positions(2,i)=0;
    plotsom(net2.layers{1}.positions)
end
```

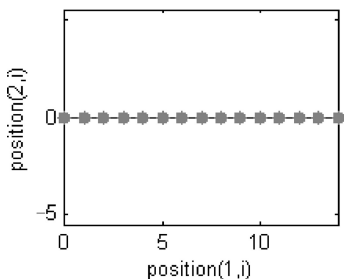


图 26-3 `net2` 第 1 网络层神经元的位置图

当然, 网络层的神经元拓扑结构还可以设置成 n 维的, 当 $n>3$ 时, 神经元的位置图将由 n 维超立体空间的点表示, 此时 `plotsom` 函数只标出开始的三维位置坐标, 图 26-4 是 `dimensions=[3 2]` 的神经元位置示意图, 它是二维的; 图 26-5 是 `dimensions=[1 3 2 1]` 的神经元位置示意图, 它是四维的, 图中只标出前三维位置坐标。

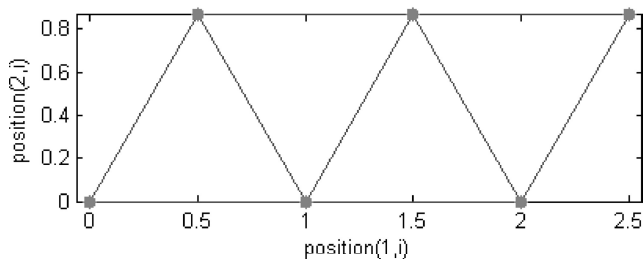


图 26-4 `dimensions=[3 2]` 的神经元位置图

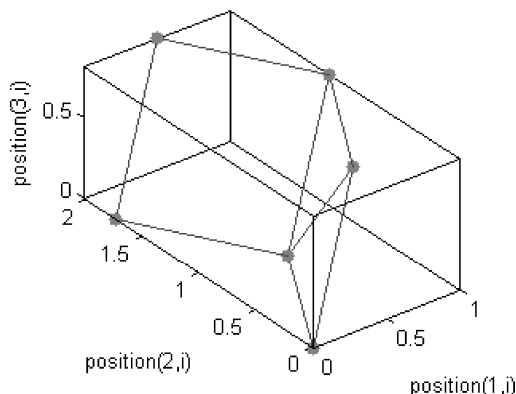


图 26-5 dimensions=[1 3 2 1]的神经元位置图

6) netInputFcn 属性

net.layers{i}.netInputFcn 属性定义一个网络输入函数，以给定的权值和阈值计算第 i 个网络层的输入。

该属性可被设置为任意一个神经网络工具箱中的网络层输入函数名（见表 26-5）。

表 26-5 神经网络工具箱的网络输入函数

函 数	说 明
netprod	求积网络输入函数
netsum	求和网络输入函数

当 sim 函数被调用时，输入函数被用于网络的仿真：

```
[Y, Pf, Af]=sim(net, P, Pi, Ai)
```

除了以上网络输入函数，用户可以自定义网络输入函数。

7) topologyFcn 属性

net.layers{i}.topologyFcn，该属性定义一拓扑结构函数，用于计算第 i 个网络层中的神经元位置（net.layers{i}.positions），该位置是通过网络层神经元的维数 net.layers{i}.dimensions 进行计算的，其值可以设置成神经网络工具箱中任意一拓扑结构函数名（见表 26-6）。

表 26-6 神经网络工具箱的拓扑结构函数

函 数	说 明
gridtop	网状网络层拓扑结构函数
hextop	六边形网络层拓扑结构函数
randtop	随机网络层拓扑结构函数

除了以上拓扑结构函数，用户可以自定义拓扑结构函数，相关内容读者可以自行参考相关文献。



TopologyFcn 属性值一旦改变, 则网络层神经元的位置 (net.layers{i}.positions) 也会随之更新。

8) size 属性

net.layers{i}.size, 该属性定义第 i 个网络层中的神经元数目, 其值可以设置为 0 或正整数。其值一旦发生变化, 则所有的 net.inputWeights{i, :}.size (连接到该网络层的输入权值向量元素的数目), net.layerWeights{:, i}.size (该网络层连接到其他网络层权值向量元素的数目), net.layerWeights{i, :}.size (其他网络层连接到该网络层权值向量元素的数目), net.biases{i}.size (该网络层阈值向量元素的数目) 等都将随之改变。

与之相关的 net.IW{i, :}, net.LW{i, :}, net.LW{:, i} 和 biases(net.b{i}) 向量的维数也会随之改变。

如果网络层具有输出向量和目标向量, 则 net.outputs{i}.size (网络层输出向量元素的数目) 和 net.targets{i}.size (目标向量元素的数目) 也会改变。

同时, 网络层神经元的维数 (net.layers{i}.dimension) 将被设置成与该属性相同的值, 这仅适用于神经元的维数为一维的情况; 对于多维的情况, 应该直接设置 net.layers{i}.dimension, 而不应使用 size 属性设置。

9) userdata 属性

net.layers{i}.userdata, 该属性值为用户提供增加关于网络层向量用户信息的地方, 它预先只定义一个字段, 其值为一段提示信息:

```
net.layers{i}.userdata=note:'Put your custom layer information here.'
```

用户可以通过修改 net.layers{i}.userdata.note 的值, 增加关于网络层的用户信息。

10) transferFcn 属性

net.layers{i}.transferFcn, 该属性定义网络的传输函数, 用于计算第 i 个网络层的输出, 该输出是通过给定的网络层输入值进行计算的。其值可以设置成神经网络工具箱中任意一个传输函数名 (见表 26-7)。

表 26-7 神经网络工具箱的传输函数

函 数	说 明
compet	竞争型传输函数
hardlim	阈值型传输函数
hardlims	对称阈值型传输函数
logsig	S 形传输函数
poslin	正线性传输函数
purelin	线性传输函数
radbas	径向基传输函数
satlin	饱和线性传输函数

续表

函 数	说 明
satlins	对称饱和线性传输函数
softmax	柔性最大值传输函数
tanhsig	双曲正切 S 形传输函数
tribas	三角形径向基传输函数

除了表中的传输函数，用户可以自定义传输函数，相关内容在后面将介绍。当 `sim` 函数被调用时，传输函数被用于网络仿真：

```
[Y, Pf, Af]=sim(net, P, Pi, Ai)
```

26.2.3 输出向量

`net.outputs`，该子对象的属性详细定义了网络的每一个输出向量，例如：

```
net1.outputs={[1×1 struct]}
net2.outputs = {[ 1×1struct]}
```

每个输出向量的结构属性值，可以通过 `net.outputs{i,j}` 访问。例如：

$$\text{net2.outputs}\{1,2\} = \begin{bmatrix} \text{size}:2 \\ \text{userdata}:[1 \times 1 \text{struct}] \end{bmatrix}$$

1) size 属性（只读）

`net.outputs{i}.size`，该属性定义了第 i 个网络层输出向量中元素的数目，其值为第 i 个网络层神经元的数目（`net.layers{i}.size`）。

2) userdata 属性

`net.outputs{i}.userdata`，该属性为用户提供了增加关于第 i 个网络层输出向量用户信息的地方，它预先只定义一个字段，其值为一段提示信息：

```
net.outputs{i}.userdata=note:'Put your custom output information here.'
```

用户可以通过修改 `net.outputs{i}.userdata.note` 的值，增加关于第 i 个网络层输出向量的用户信息。

26.2.4 阈值向量

`net.biases`，该子对象的属性详细定义网络的每一个阈值向量。例如：

```
net1.biases = {[1×1struct]}
net2.biases =  $\begin{bmatrix} [1 \times 1 \text{struct}] \\ [1 \times 1 \text{struct}] \end{bmatrix}$ 
```



每个阈值向量的结构属性值可以通过 `net.biases{i}` 访问。例如：

```
net2.biases{1} =
    initFcn : "
    learn : 1
    learnFcn : 'learngdm'
    learnParam : [1×1 struct]
    size : 15
    userdata : [[1×1 struct]]
```

1) initFcn 属性

`net.biases{i}.initFcn`，该属性定义第 i 个网络层阈值向量的初始化函数，如果网络的初始化函数为 `initlay`，则第 i 个网络层阈值向量的初始化函数为 `initwb`，其值可以是设置成神经网络工具箱中任意一个阈值初始化函数名（见表 26-8）。

表 26-8 神经网络工具箱的初始化函数

函 数	说 明
initcon	"良心"（conscience）阈值初始化函数
initzero	零权值/阈值初始化函数
rands	对称随机权值/阈值初始化函数

除了表中的初始化函数，用户还可以自定义初始化函数，当 `init` 函数被调用时，`initFcn` 定义的函数将对第 i 个网络层阈值向量（`net.b{i}`）进行初始化：

```
net=init(net)
```

2) learn 属性

`net.biases{i}.learn`，该属性定义第 i 个阈值向量在训练和调整过程中是否变化。其值可以设置为 0 或 1。它容许或禁止在训练和调整过程中对阈值向量进行学习：

```
[net,Y,E,Pf,aF]=adapt(NET,P,T,Pi,Ai)
[net,tr]=train(NET,P,T,Pi,Ai)
```

3) learnParam 属性

`net.biases{i}.learnParam`，该属性定义了第 i 个网络层阈值向量当前学习函数的参数及参数值，其值取决于当前的学习函数（`net.biases{i}.learnFcn`），可以查阅当前学习函数的参考文献帮助获得当前学习函数的具体描述：

```
help(net.biases{i}.learnFcn)
```

4) learnFcn 属性

`net.baíses{i}.learnFcn`，如果网络的训练函数是 `trainb`，`trainc` 和 `trainr`，或者网络调整函数为 `trains`，则该属性定义第 i 个网络层阈值向量在训练和调整过程中的学习函数，其值可以设

置成神经网络工具箱中任意一个学习函数名（见表 26-9）。

表 26-9 神经网络工具箱的学习函数

函 数	说 明
learncon	"良心" (conscience) 阈值学习函数
learngd	梯度下降权值/阈值学习函数
learnghm	附加动量因子的梯度下降权值/阈值学习函数
learnp	感知器权值/阈值学习函数
learnpn	归一化感知器权值/阈值学习函数
learnwh	WH (Widrow-Hoff) 权值/阈值学习规则

除了以上学习函数，用户还可以自定义学习函数。如果网络的训练函数是 `trainb`，`trainc` 和 `trainr`，或者网络调整函数为 `trains`，则当 `train` 函数被调用时，`learnFcn` 定义的函数将对第 i 个网络层阈值向量 (`net.b{i}`) 进行更新：

```
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)
[net,tr]=train(NET, P, T, Pi, Ai)
```

该属性值一旦改变，则阈值向量的学习参数 (`net.biases{i}.learnFcn`) 将被新定义的学习函数的参数及其默认参数值代替。

5) size 属性（只读）

`net.biases{i}.size`，该属性定义了第 i 个网络层阈值向量元素的数目，其值为第 i 个网络层神经元的数目 (`net.layers{i}.size`)。

6) usersdata 属性

`net.biases{i}.usersdata`，该属性为用户提供了增加关于第 i 个网络层阈值向量的用户信息，可以通过修改 `net.biases{i}.usersdata.note` 的值增加关于第 i 个阈值向量的用户信息。

26.2.5 输入权值向量

`net.inputWeights`，该子对象的属性详细定义网络的每一个输入权值向量。例如：

```
net1.inputWeights = {[1×1struct]}
```

```
net2.inputWeights = {[1×1struct]
                     []}
```

每个输入权值向量的结构属性值可以通过 `net.inputWeights{i,j}` 访问。例如：



```
net1.inputWeights{1} =
    delays : 0
    initFcn : 'initzero'
    learn : 1
    learnFcn : 'learnp'
    learnParam : []
    size : [2 3]
    userdata : [1x1 struct]
    weightFcn : 'dotprod'
```

1) delays 属性

`net.inputWeights{i,j}.delays`, 该属性定义第 j 个输入向量与第 i 个网络层权值之间的抽头延迟线, 其值为从 0 或正整数开始逐渐增大的行向量。

该属性值一旦改变, 权值向量的大小 (`net.inputWeights{i,j}.size`) 和权值向量维数 (`net.IW{i,j}`) 将被更新。

2) initFcn 属性

`net.inputWeights{i,j}.initFcn`, 如果网络的初始化函数为 `initlay`, 且网络层的初始化函数为 `initwb`, 则该属性定义第 j 个输入向量与第 i 个网络层权值向量的初始化函数, 其值可以设置成神经网络工具箱中任意一个权值初始化函数名 (如表 26-10 所示)。

表 26-10 神经网络工具箱的权值初始化函数

函 数	说 明
<code>initzero</code>	零权值/阈值初始化函数
<code>midpoint</code>	中点权值初始化函数
<code>randnc</code>	归一化列权值初始化函数
<code>randnr</code>	归一化行权值初始化函数
<code>rands</code>	对称随机权值/阈值初始化函数

除了以上权值初始化函数, 用户可以自定义权值初始化函数。如果网络的初始化函数 (`net.initFcn`) 为 `initlay`, 且网络层的初始化函数 (`net.layers{i}.initFcn`) 为 `initwb`, 则当 `init` 函数被调用时, `net.inputWeights{i,j}.initFcn` 所定义的函数将用于计算从第 j 个输入向量到第 i 个网络层的初始权值向量值:

```
net=init(net)
```

3) learn 属性

`net.inputWeights{i,j}.learn`, 该属性定义第 j 个输入向量与第 i 个网络层的权值向量在训练和调整过程中是否改变, 其值可设置为 0 或 1, 分别表示权值向量在训练和调整的过程中禁止和容许进行学习:



```
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)
[net,tr]=train(NET, P, T, Pi, Ai)
```

4) learnFcn 属性

`net.inputWeights{i,j}.learnFcn`, 如果网络的训练函数为 `trainb`, `trainc` 和 `trainr`, 或网络的调整函数为 `trains`, 则该属性定义第 j 个输入向量与第 i 个网络层权值向量的学习函数, 其值可以设置成神经网络工具箱中任意一个权值学习函数名 (见表 26-11)。

表 26-11 神经网络工具箱的权值学习函数

函 数	说 明
<code>learngd</code>	梯度下降权值/阈值学习函数
<code>learnghdm</code>	附加动量因子的梯度下降权值/阈值学习函数
<code>learnh</code>	Hebb 权值学习函数
<code>learnhhd</code>	带衰减因子的 Hebb 权值学习函数
<code>learnis</code>	内星权值学习函数
<code>learnk</code>	Kohonen 权值学习函数
<code>learnlv1</code>	LVQ1 权值学习函数
<code>learnlv2</code>	LVQ2 权值学习函数
<code>learnos</code>	外星权值学习函数
<code>learnp</code>	感知器权值/阈值学习函数
<code>learnpn</code>	归一化感知器权值/阈值学习函数
<code>learnsom</code>	自组织映射权值学习函数
<code>learnwh</code>	WH(Widrow-Hoff)权值/阈值学习规则

除了以上权值初始化函数, 用户还可以自定义权值初始化函数。如果网络的训练函数 (`net.trainFcn`) 为 `trainb`, `trainc` 和 `trainr`, 或网络的调整函数 (`net.adaptFcn`) 为 `trains`, 则 i 当 `train` 函数被调用时, `net.inputWeights{i,j}.learn` 所定义的函数将用于计算从第 j 个输入向量与第 i 个网络层的权值向量:

```
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)
[net,tr]=train(NET, P, T, Pi, Ai)
```

5) learnParam 属性

`net.inputWeights{i,j}.learnParam`, 该属性定义第 i 个网络层来自第 j 个输入向量的权值向量学习函数的参数及参数值, 其值取决于当前的学习函数 (`net.inputWeights{i,j}.learnFcn`), 可以查阅当前学习函数的帮助以获得当前学习函数的具体描述:

```
help(net.inputWeights{i,j}.learnFcn)
```



6) size 属性（只读）

`net.inputWeights{i,j}.size`，该属性定义第 i 个网络层与第 j 个输入向量连续权的数目，其值为具有两个元素的行向量，分别表示相应权值向量（`net.IW{i,j}`）的行数与列数。第一个元素的值为第 i 个网络层的神经元的数目（`net.layers{i}.size`），第二个元素的值为第 j 个输入向量元素的数目（`net.inputs{j}.size`）与第 i 个网络层输入抽头延迟线长度的乘积：

```
length(net.inputWeights{i,j}.delays)*net.inputs{j}.size
```

7) userdata 属性

`net.inputWeights{i,j}.userdata`，该属性为用户提供了增加关于第 i 个网络层与第 j 个输入向量连接权数目用户信息的地方，它预先只定义一个字段，其值为一段提示信息：

```
net.inputWeights{i,j}.userdata.note:'Put your custom weight information here.'
```

用户可以通过修改 `net.inputWeights{i,j}.userdata.note` 的值，增加关于第 i 个网络层与第 j 个输入向量连接权数目的用户信息。

8) weightFcn 属性

`net.inputWeights{i,j}.weightFcn`，该属性定义第 i 个网络层来自第 j 个输入向量的权值函数，其值可以是神经网络工具箱的任意一个权值函数名（见表 26-12）。

表 26-12 神经网络工具箱的权值函数

函 数	说 明
dist	欧几里得（Euclidean）距离权值函数
dotprod	点积权值函数
mandist	曼哈顿（Manhattan）距离权值函数
negdist	归一化列权值初始化函数
normprod	归一化行权值初始化函数

当 `sim` 函数被调用时，权值函数被用于网络的仿真：

```
[Y, Pf, Af]=sim(net, P, Pi, Ai)
```

26.2.6 目标向量

`net.targets`，该子对象的属性详细定义了网络的每一个目标向量。例如：

```
net.targets={ [1×1 struct] }  
net2.targets = [ ] [1×1 struct]
```

每个目标向量的结构属性值可以通过 `net.targets{i,j}` 访问。例如：

$$\text{net2.targets}\{1,2\} = \begin{cases} \text{size}: 2 \\ \text{userdata}: [1 \times 1 \text{struct}] \end{cases}$$

1) size 属性 (只读)

`net.targets{i}.size`, 该属性定义了第 i 个网络层目标向量中元素的数目, 其值为第 i 个网络层神经元的数目 (`net.layers{i}.size`)。

2) userdata 属性

`net.targets{i}.userdata`, 该属性为用户提供了增加关于第 i 个网络层目标向量用户信息的地方, 它预先只定义一个字段, 其值为一段提示信息:

```
net.targets{i}.userdata.note='Put your custom targets information here.'
```

用户可以通过修改 `net.targets{i}.userdata.note` 的值, 增加关于第 i 个目标向量的用户信息。

26.2.7 网络层权值向量

```
net.layerWeights={[]}  
net2.layerWeights = {  
    []  
    [1x1struct] []  
}
```

每个网络层权值向量的结构属性值可以通过 `net.layerWeights{i,j}` 访问。例如:

$$\text{net2.layerWeights}\{2,1\} = \begin{bmatrix} \text{delays}: 0 \\ \text{initFcn}: '' \\ \text{learn}: 1 \\ \text{learnFcn}: 'learn\text{gdm}' \\ \text{learnParam}: [1 \times 1 \text{struct}] \\ \text{size}[2 \quad 15] \\ \text{userdata}: [1 \times 1 \text{struct}] \\ \text{weightFcn}: 'dotprod' \end{bmatrix}$$

1) delays 属性

`net.layersWeights{i,j}.delays`, 该属性定义第 j 个网络层与第 i 个网络层权值之间的抽头延迟线, 其值为从 0 或正整数开始的逐渐增大的行向量。

2) initFcn 属性

`net.layerWeights{i,j}.initFcn`, 如果网络的初始化函数为 `initlay`, 且网络层的初始化函数为 `initwb`, 则该属性定义从第 j 个网络层到 i 个网络层权值向量的初始化函数, 其值可以设置成神经网络工具箱中任意一个权值初始化函数名 (见表 26-10)。

如果网络的初始化函数 (`net.initFcn`) 为 `initlay`, 且网络层的初始化函数 (`net.layers{i}.initFcn`) 为 `initwb`, 则当 `init` 函数被调用时, `net.layerWeights{i,j}.initFcn` 所定义的函数将用于



计算从第 j 个网络层到第 i 个网络层的初始权值向量值:

```
net=init(net)
```

3) learn 属性

`net.layerWeights{i,j}.learn`, 该属性定义第 j 个网络层到第 i 个网络层的权值向量在训练和调整过程中是否改变, 其值可设置为 0 或 1, 分别表示权值向量在训练和调整过程中禁止和容许进行学习。

```
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)
[net,tr]=train(NET, P, T, Pi, Ai)
```

4) learnParam 属性

`net.layerWeights{i,j}.learnParam`, 该属性定义第 i 个网络层来自第 j 个网络层的权值向量学习函数的参数及参数值, 其值取决于当前的学习函数 (`net.layerWeights{i,j}.learnFcn`), 可以查阅当前学习函数的帮助, 获得当前学习函数的具体描述:

```
help(net.layerWeights{i,j}.learnFcn)
```

5) learnFcn 属性

`net.layerWeights{i,j}.learnFcn`, 如果网络的训练函数为 `trainb`, `trainc` 和 `trainr`, 或网络的调整函数为 `trains`, 则该属性定义第 j 个网络层到第 i 个网络层权值向量的学习函数, 其值可以设置成神经网络工具箱中任意一个权值学习函数名 (见表 26-11)。如果网络的训练函数 (`net.trainFcn`) 为 `trainb`, `trainc` 和 `trainr`, 或网络的调整函数 (`net.adaptFcn`) 为 `trains`, 则当 `train` 函数被调用时, `net.layerWeights{i,j}.learnFcn` 所定义的函数将用于计算从第 j 个网络层到第 i 个网络层的权值向量:

```
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)
[net, tr]=train(NET, P, T, Pi, Ai)
```

6) size 属性 (只读)

`net.layerWeights{i,j}.size`, 该属性定义第 i 个网络层来自第 j 个网络层的权值向量元素的数目, 其值为具有两个元素的行向量, 分别表示相应网络层权值向量 (`net.LW{i,j}`) 的行数与列数。第一个元素的值为第 i 个网络层神经元的数目 (`net.layers{i}.size`), 第二个元素的值为第 j 个网络层神经元的数目 (`net.layers{j}.size`) 与权值抽头延迟线长度的乘积:

```
length(net.layerWeights{i,j}delays)*net.layers{j}.size
```

7) userdata 属性

`net.layerWeights{i,j}.userdata`, 该属性为用户提供增加关于第 i 个网络层来自第 j 个网络层的权值向量元素数目用户信息的地方, 它预先只定义一个字段, 其值为一段提示信息:

```
net.layerWeights{i,j}.userdata=note:'Put your custom weight information here.'
```

用户可以通过修改 `net.layerWeights{i,j}.userdata.note` 的值, 增加关于第 i 个网络层来自第 j 个

输入向量的权值向量元素数目的用户信息。

8) weightFcn 属性

`net.layerWeights{i,j}.weightFcn`, 该属性定义第 i 个网络层来自第 j 个网络层的权值函数, 其值可以是神经网络工具箱的任意一个权值函数名 (如表 26-12 所示)。

当 `sim` 函数被调用时, 权值函数被用于网络的仿真:

`[Y, Pf, Af]=sim(net, P, Pi, Ai)`

以上介绍了 MATLAB 神经网络工具箱中神经元和神经网络的一般模型、神经网络对象及其子对象的属性, 理解这些内容, 将有助于我们利用神经网络工具箱进行神经网络的设计和仿真。

通过下面几个例子, 读者可以进一步熟悉本章的基本内容。

【例 26-1】 设 $p = [2 \quad -2]^T$, $IW^1 = \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & -0.1 \\ 0.9 & -0.0 \end{bmatrix}$, $b^1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.4 \end{bmatrix}$, $f^1 = \begin{bmatrix} \text{purelin} \\ \text{purelin} \\ \text{purelin} \end{bmatrix}$, 试画出

其网络结构示意图, 并求网络的输出值。

解析: 根据题意, 神经网络有 1 个输入向量, 包含 2 个输入变量; 输入层有 3 个神经元, 传输函数为 `purelin`; 无其他网络层, 所以为单层神经网络。根据以上可画出其神经网络结构示意图, 如图 26-6 所示。

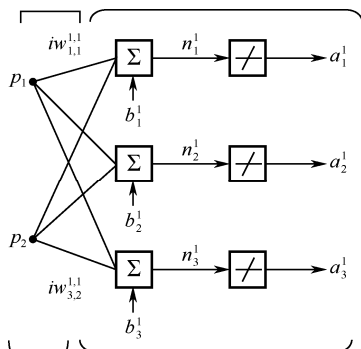


图 26-6 例 26-1 的神经网络结构图

则

$$a^1 = f^1(IW^1 p + b^1) = f^1 \begin{bmatrix} 0.5 & 0.5 \\ 0.9 & -0.1 \\ 0.9 & 0.1 \end{bmatrix} \begin{bmatrix} 2 \\ -2 \end{bmatrix} + \begin{bmatrix} 0.5 \\ -0.5 \\ 0.4 \end{bmatrix} = \begin{bmatrix} \text{purelin}(0.5) \\ \text{purelin}(1.5) \\ \text{purelin}(2.0) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.5 \\ 2.0 \end{bmatrix}$$

【例 26-2】 某个神经网络的部分属性如下, 据此画出其网络向量模型结构示意图。

```
net.numInputs=1
net.inputs{1}.size=2
net.numLayers=2
net.layers{1}.dimensions=3
net.layers{2}.dimensions=2
```



```
net.biasConnect=[1;1]
net.inputConnect=[1;0]
net.layerConnect=[0 0;1 0]
net.outputConnect=[0 1]
net.outputs{1}.size=2
net.layers{1}.transferFcn=logsig
net.layers{2}.transferFcn=logsig
```

解析: 根据题意, 网络有 1 个输入向量 ($\text{net.numInputs}=1$), 包含 2 个输入变量 ($\text{net.inputs}\{1\}.\text{size}=2$); 有 2 个网络层 ($\text{net.numLayers}=2$), 第 1 个网络层有 3 个神经元 ($\text{net.layers}\{1\}.\text{dimensions}=3$), 第 2 个网络层有 2 个神经元 ($\text{net.layers}\{2\}.\text{dimensions}=2$), 每个网络层的神经元都有阈值 ($\text{net.biasConnect}=[1;1]$); 第 1 个网络层 (输入层) 与输入向量连接, 第 2 个网络层与输入向量无连接 ($\text{net.inputConnect}=[1;0]$); 只有第 1 个网络层到第 2 个网络层的连接, 无其他网络层连接 ($\text{net.layerConnect}=[0 \ 0;1 \ 0]$); 两层网络层神经元的传输函数均为 logsig ($\text{net.layers}\{1\}.\text{transferFcn}=\text{logsig}$, $\text{net.layers}\{2\}.\text{transferFcn}=\text{logsig}$); 第 2 个网络层为输出层, 有 2 个输出变量 ($\text{net.outputConnect}=[0 \ 1]$, $\text{net.outputs}\{1\}.\text{size}=2$)。据以上所述画出网络向量模型结构示意图, 如图 26-7 所示。

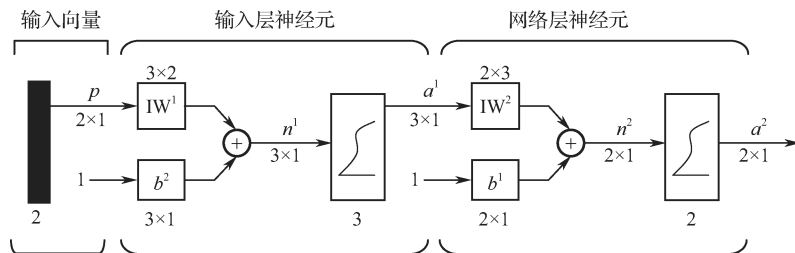
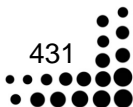


图 26-7 例 26-2 神经网络的向量模型结构示意图



第 27 章 自定义函数及其应用

神经网络工具箱允许创建并应用很多种类的函数，用户可以根据需要，在初始化、仿真及训练中应用多种方法，并实现对网络的自行调整，这些自行编制的函数称为自定义函数。

本章介绍如何创建自定义函数，这些函数主要用来进行神经网络的初始化、学习、训练和仿真，可以大致分为四类。

(1) 初始化函数

- 网络初始化函数
- 层初始化函数
- 权值和阈值初始化函数

(2) 学习函数

- 网络训练函数
- 网络自适应函数
- 网络性能函数
- 权值和阈值学习函数

(3) 仿真函数

- 传递函数
- 传递函数导函数
- 网络输入函数
- 网络输入函数导函数
- 权值函数
- 权值导函数

(4) 自组织映射函数

- 拓扑函数
- 距离函数

27.1 初始化函数

初始化函数包括网络、层、权值和阈值三类初始化函数，下面分别介绍如何自定义生成这三种函数。

1. 网络初始化函数

网络初始化函数将所有的权值和阈值设置为一个适当的值，作为网络训练或者自适应调



节的初始点。一旦定义了网络初始化函数，就可以嵌入某一网络中。假设定义网络初始化函数为 `xiu()`，并嵌入某一网络中，则应用下面语句实现。

```
net.initFcn='xiu'
```

这样，在调用 `init()` 初始化网络时，都可以应用此时设定的网络初始化函数进行初始化。

```
net=init (net)
```

网络初始化函数编制完成后，接受某一网络，并且在初始化处理以后，再返回一个网络。

```
net=xiu (net, i)
```

自定义网络初始化函数可以根据要求对权值和阈值进行任意设置。

2. 层初始化函数

层初始化函数将某层所有的权值和阈值设置为一个适当的值，作为网络训练或者自适应调节的初始点。一旦定义了层初始化函数，就可以嵌入网络任意一层上。假设定义了层初始化函数为 `xiu()`，并嵌入网络第三层上，则应用上面语句实现。

```
net.layers{3}.initFcn='xiu'
```

如果网络初始化函数 (`net.initFcn`) 设置为工具箱函数 `initlay()`，自定义层初始化函数用来对层进行初始化。那么在调用 `init()` 初始化网络时，都可以应用此时设定的层初始化函数进行初始化。

```
net=init (net)
```

层初始化函数编制完成后，接收网络和层的标号作为输入 `i`，并且在对第 `i` 层初始化处理以后，再返回网络。

```
net=xiu (net, i)
```

自定义层初始化函数可以根据要求对层的权值和阈值进行任意设置。

3. 权值和阈值初始化函数

权值和阈值初始化函数将所有的权值和阈值设置为一个适当的值，作为网络训练或者自适应调节的初始点。一旦定义了权值和阈值初始化函数，就可以嵌入网络任意权值和阈值上。假设定义了权值和阈值初始化函数为 `xium()`，并嵌入网络第二层的阈值、第一层输入到第二层的权值上，则应用下面的语句实现。

```
net=init (net)
```

权值和阈值初始化函数编制完成后，可以应用如下方式进行调用：

```
W=randS (S, PR)
```

```
b=randS (S)
```

其中，`S` 为层神经元数目；`PR` 为 `R` 个输入向量的最大/最小值矩阵。

神经网络工具箱中包含一个自定义权值和阈值初始化函数 `mywbif()`，输入 `help myxiu` 就可以获得有关此函数的帮助信息。下面举例说明如何应用 `myxiu()` 进行权值和阈值的初始化。



【例 27-1】

```
w=mywbif(4,[0 2;-2 3])
b=mywbif(4,[1 1])
```

输出结果为:

```
w =
    0.0173    0.0876
    0.0980    0.0737
    0.0271    0.0137
    0.0252    0.0012
b =
    0.0894
    0.0199
    0.0299
    0.0661
```

输入 type mywbif, 可以查看 mywbif()的源程序。

```
>> type mywbif
function w = mywbif(s,pr)
%MYWBIF Example custom weight and bias initialization function.
%
% Use this function as a template to write your own function.
%
% Syntax
%
% W = rands(S,PR)
% S - number of neurons.
% PR - Rx2 matrix of R input ranges.
% W - SxR weight matrix.
%
% b = rands(S)
% S - number of neurons.
% b - Sx1 bias vector.
%
% Example
%
% W = mywbif(4,[0 1;-2 2])
% b = mywbif(4,[1 1])
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

if nargin < 1, error('Not enough input arguments'), end
if nargin == 1
```



```
w = rand(s,1)*0.2; % <-- Replace with your own initial bias vector
else
    r = size(pr,1);      % <-- Replace with your own initial weight matrix
    w = rand(s,r)*0.1;
end
```

可以将函数 `mywbif()` 作为一个模板，用来生成自定义的权值和阈值初始化函数。

27.2 学习函数

与网络学习、权值和阈值调整有关系的学习函数有四类：训练函数、自适应函数、性能函数、权值和阈值调整函数，下面分别介绍如何自定义生成这些函数。

1. 训练函数

训练函数是常用的学习函数。学习函数循环地将输入向量应用于网络中，每次都能够更新网络，直到达到训练目标位置。训练停止条件可以是最大学习次数、最小的误差梯度或者训练精度等。一旦定义了训练函数，就可以嵌入某一网络中。假设定义了训练函数为 `xiu()`，并嵌入某个网络中，则应用下面语句实现。

```
net.trainFcn='xiu'
```

这样，训练网络时，都可以应用此时设定的训练函数。

```
[net, tr]=train (NET, P, T, Pi, Ai)
```

训练函数编制完成后，可以应用如下方式进行调用：

```
[net, tr]=xiu (net, Pd, T1, Ai, Q, TS, VV, TV)
```

自定义训练函数需要提供如下信息：

- **version**：神经网络工具箱的版本。
- **pdefaults**：默认参数。

自定义训练函数可以根据所设定的任意方式更新网络的权值和阈值。

2. 自适应函数

自适应函数在每个输入时间段内都要更新网络，并进行仿真。一旦定义了自适应函数，就可以嵌入某一网络上。假设定义了自适应函数为 `xium()`，并嵌入了某个网络中，则应用下面语句实现。

```
net.adaptFcn='xium';
```

这样，自适应调节网络时，都可以应用此时设定的自适应函数。

```
[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)
```

自适应函数编制完成后，可以应用如下方式进行调用：

```
[net, Ac, E1]=xium (net, Pd, Ti, Ai, Q, TS)
```



自定义自适应函数需要提供如下信息：

- version：神经网络工具箱的版本。
- pdefaults：默认参数。

自定义自适应函数可以根据所设定的任意方式更新网络的权值和阈值。

3. 性能函数

性能函数是学习训练时期望达到最优的指标，通过改变权值和阈值使性能函数最优，改善网络性能。一旦定义了性能函数，就可以嵌入某一网络中。假设定义性能函数为 `xiuw()`，并嵌入某个网络中，则应用下面语句实现。

```
net.performFcn='xiuw'
```

这样，训练或者自适应调节网络时，都可以应用此时设定的性能函数进行优化。

```
[net, tr]=train (NET, P, T, Pi, Ai)
```

```
[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)
```

性能函数编制完成后，可以应用如下方式进行调用：

```
perf=xiuw (E, X, PP)
```

其中，`E` 为期望值矩阵；`X` 为网络权值和阈值；`PP` 为网络参数。

如果 `E` 是细胞数组，则需要首先转换成矩阵形式，然后再进行调用，`X` 和 `PP` 的值通过网络得到：

```
E=cell2mat (E);  
perf=xiuw (E, net)  
X=getx (net);  
PP=net.performParam;
```

自定义性能函数需要提供如下信息：

- version：神经网络工具箱的版本。
- deriv：相关导数函数名。
- pdefaults：默认参数。

神经网络工具箱中包含一个自定义性能函数 `myperf()`，输入 `help myperf` 就可以获得有关此函数的帮助信息。下面举例说明如何应用 `myperf()` 性能函数，随机输入网络权值和阈值，并应用网络默认参数计算相应的性能函数值。

【例 27-2】

```
e=rand(4,5);  
x=rand(12,1);  
pp=myperf('pdefaults')  
perf=myperf(e,x,pp)
```

输入结果为：

```
pp =  
x: 1
```



```
y: 0.5000
perf =
    0.7721
```

输入 `type mypf`, 可以查看 `mypf()` 的源程序。

```
>> type mypf
function perf = mypf(e,x,pp)
%MYPF Example custom performance function.
%
%   Use this function as a template to write your own function.
%
%   Calculation Syntax
%
%       perf = mypf(E,X,PP)
%       E   - Matrix or cell array of error vector(s).
%       X   - Vector of all weight and bias values.
%       PP  - Performance parameter.
%
%       perf = mypf(E,net)
%
%   Information Syntax
%
%       info = mytf(code) returns useful information for each CODE string:
%       'version' - Returns the Neural Network Toolbox version (3.0).
%       'deriv'   - Returns the name of the associated derivative function.
%       'output'  - Returns the output range.
%       'active'  - Returns the active input range.
%
%   Example
%
%       e = rand(4,5);
%       x = rand(12,1);
%       pp = mypf('pdefaults')
%       perf = mypf(e,x,pp)
% Copyright 1997-2001 The MathWorks, Inc.
% $Revision: 1.4.2.1 $
if nargin < 1, error('Not enough arguments.');
```

```
end
if isstr(e)
    switch (e)
    case 'version'
        perf = 3.0;          % <-- Must be 3.0.
    case 'deriv',
        perf = 'mydpf';      % <-- Replace with the name of your derivative function or "
```





```

case 'name',
    perf = 'Custom';    % <-- Replace with your function's name
case 'pnames',
    perf = {};          % <-- Add names of your function parameters (if any)
case 'pdefaults'
    perf.x = 1;         % <-- Replace with the your own performance
    perf.y = 0.5;       % <-- parameter structure or null matrix [ ].
otherwise, error('Unrecognized code.')
end
else
    if isa(e,'cell')
        e = cell2mat(e);
    end
    if nargin == 2
        pp = x.performParam; % <-- delete this line if you don't use PP
        x = getx(net);       % <-- delete this line if you don't use X
    end
    % ** Replace the following calculation with your own
    % ** measure of performance.
    numErrors = prod(size(e));
    numWeightsBiases = length(x);
    perf = sum(sum(abs(e))) * pp.x/numErrors + ...
           sum(abs(x)) * pp.y/numWeightsBiases;
end
end

```

4. 权值和阈值学习函数

权值和阈值学习函数与某些训练或者自适应调节函数一起使用，用于学习训练中更新权值和阈值。一旦定义了权值和阈值学习函数，就可以嵌入网络任意一层上。假设定义了权值和阈值学习函数 `xiu()`，并嵌入网络第二层，则应用下面语句实现。

```
net.layers{2}.learnFcn='xiu';
```

这样，如果网络训练函数（`net.trainFcn`）设置为 `trainb`、`trainc` 或者 `trainr`，网络自适应调节函数（`net.adaptFcn`）设置为 `trains`，都可以应用此时设定的权值和阈值学习函数更新权值和阈值。

```

[net, tr]=train (NET, P, T, Pi, Ai)
[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)

```

权值和阈值学习函数编制完成后，可以应用如下的方式进行调用，以更新权值和阈值：

```

[dW, LS]=xiu (W, P, Z, N, A, T, E, gW, gA, D, LP, LS)
[db, LS]=xiu (b, ones (1, Q), Z, N, A, T, E, gW, gA, D, LP, LS)

```

自定义权值和阈值学习函数需要提供如下信息：



- version: 神经网络工具箱的版本。
- deriv: 相关导数函数名。
- pdefaults: 默认参数。

神经网络工具箱中包含一个自定义权值和阈值学习函数 mywblf(), 输入 help mywblf 就可以获得有关此函数的帮助信息。

```
>> type mywblf
function [dw,ls] = mywblf(w,p,z,n,a,t,e,gW,gA,d,lp,ls)
%MYWBLF Example custom weight and bias learning function.
%
% Calculation Syntax
%
% [dW,LS] = mywblf(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
% [db,LS] = mywblf(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
% W - SxR weight matrix (or Sx1 bias vector).
% P - RxQ input vectors (or ones(1,Q)).
% Z - SxQ weighted input vectors.
% N - SxQ net input vectors.
% A - SxQ output vectors.
% T - SxQ layer target vectors.
% E - SxQ layer error vectors.
% gW - SxR gradient with respect to performance.
% gA - SxQ output gradient with respect to performance.
% D - SxS neuron distances.
% LP - Learning parameters, none, LP = [].
% LS - Learning state, initially should be = [].
% dw - SxR weight (or bias) change matrix.
%
% Information Syntax
%
% info = mywblf(code) returns useful information for each CODE string:
% 'version' - Returns the Neural Network Toolbox version (3.0).
% 'pdefaults' - Returns the name of the associated derivative function.
% 'needg' - Returns the output range.
%
% Example
%
% W = rand(4,5);
% gW = rand(4,5);
% lp = mywblf('pdefaults')
% [dW,ls] = mywblf(w,[],[],[],[],[],gW,[],[],lp,[]);
% W = W + dW;
```



```

%   gW = rand(4,5);
%   [dW,ls] = mywblf(w,[],[],[],[],gW,[],[],lp,ls);
%   W = W + dW;
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.3.2.1 $
if isstr(w)
    switch lower(w)
        case 'version'
            dw = 3.0;          % <-- Must be 3.0.

        case 'pdefaults'
            dw.lr = 0.01;      % <-- Replace with your own learning
                                %      parameters or the null matrix [].

        case 'needg'
            dw = 1;           % <-- 1 or 0 depending on whether your
                                %      function uses gW or gA, or not.

        otherwise
            error('Unrecognized property.')
        end
    end
else
    if isempty(ls)
        ls.x = 0.3;           % <-- Replace with your own functions initial
                                %      learning state or the null matrix []
    end
    dw = lp.lr*ls.x*gW; % <-- Replace with your own weight change
                                %      calculation.
    ls.x = 1-ls.x;           % <-- Replace with your own learning state
                                %      update code, if you have any such state.
end
end

```

27.3 仿真函数

自定义仿真函数包括传递函数、网络输入函数、权值函数。下面分别介绍如何创建这三种仿真函数及其相应的导数函数。

27.3.1 传递函数

传递函数根据给定的网络输入向量（或者矩阵） N ，计算某层的输出向量（或矩阵） A ，网络输入向量和输出向量必须具有相同的维数。一旦定义了传递函数，就可以嵌入网络中任意一层中。假设定义了传递函数为 `xiux()`，并嵌入网络第三层，则应用下面语句实现：



```
net.layers{3}.transferFcn='xiux'
```

这样，在对网络进行仿真时，都可以应用此时设定的传递函数。

```
[Y, Pf, Af]=sim (net, P, Pi, Ai)
```

传递函数编制完成后，可以应用如下方式进行调用：

```
A=xiux (N)
```

其中，N 为网络输入向量；A 为函数返回值，即网络输出向量。

自定义传递函数返回如下信息：

- version: 神经网络工具箱的版本。
- deriv: 相关导数函数名。
- output: 输出范围。
- active: 活动的输入范围。

神经网络工具箱中包含了一个自定义传递函数 mytf(), 输入 help mytf 就可以获得有关此函数的帮助信息。

【例 27-3】 绘制传递函数 mytf() 曲线，如图 27-1 所示。

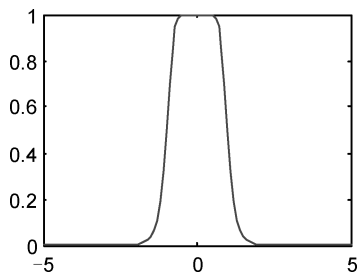
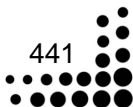


图 27-1 自定义传递函数 mytf() 曲线

```
N=-5:0.1:5;
A=mytf(N);
plot(N,A)
mytf('deriv')
ans =
mydtf
```

输入 type mytf，可以查看 mytf() 的源程序。

```
>> type mytf
function a = mytf(n)
%MYTF Example custom transfer function.
%
% Use this function as a template to write your own function.
%
% Calculation Syntax
%
```





```
% A = mytf(N)
% N - SxQ matrix of Q net input (column) vectors.
% A - SxQ matrix of Q output (column) vectors.
%
% Information Syntax
%
% info = mytf(code) returns useful information for each CODE string:
% 'version' - Returns the Neural Network Toolbox version (3.0).
% 'deriv' - Returns the name of the associated derivative function.
% 'output' - Returns the output range.
% 'active' - Returns the active input range.
%
% Example
%
% n = -5:1:5;
% a = mytf(n);
% plot(n,a)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
if nargin < 1, error('Not enough arguments.');
```

```
end
if isstr(n)
    switch (n)
        case 'version'
            a = 3.0; % <-- Must be 3.0.
        case 'deriv'
            a = 'mydtf'; % <-- Replace with the name of your
                        % associated function or "
        case 'output'
            a = [-1 1]; % <-- Replace with the minimum and maximum
                        % output values of your transfer function
        case 'active'
            a = [-2 2]; % <-- Replace with the range of inputs where
                        % the outputs are most sensitive to changes.
            otherwise, error('Unrecognized code.')
```

```
end
else
    a = 1./(n.^8+1); % <-- Replace with your calculation
end
```

可以将函数 mytf() 作为一个模板，用来生成自定义的传递函数。



27.3.2 传递函数导数函数

如果自定义传递函数需要回传，则需要自定义传递函数导数函数，传递函数导数函数根据给定的网络输入来计算此层输出的导数。假设定义传递函数导数函数为 `xiudx()`，则应用下面语句计算此层输出的导数。

```
dA_dN=xiudx(N,A)
```

其中，`N` 为网络输入向量；`A` 为网络输出向量；`dA_dN` 表示导数 dA/dN 。

神经网络工具箱中包含了一个自定义传递函数导数函数 `mydtf()`，输入 `help mydtf` 就可以获得有关此函数的帮助信息。

【例 27-4】 绘制传递函数导数函数曲线，如图 27-2 所示。

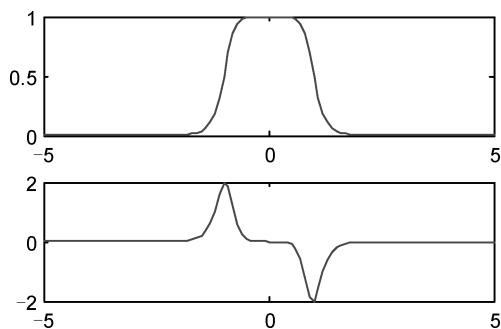


图 27-2 自定义传递函数导数函数 `mydtf()` 曲线

```
N=-5:0.1:5;
A=mytf(N);
dA_dN=mydtf(N,A);
subplot(211)
plot(N,A)
subplot(212)
plot(N,dA_dN)
```

输入 `type mydtf`，可以查看 `mydtf()` 的源程序。

```
>> type mydtf
function d = mydtf(n,a)
%MYDTF Example custom transfer derivative function of MYTF.
%
% Use this function as a template to write your own function.
%
% Syntax
%
% dA_dN = mydtf(N,A)
% N - SxQ matrix of Q net input (column) vectors.
```





```

%      A - SxQ matrix of Q output (column) vectors.
%      dA_dN - SxQ derivative dA/dN.
%
% Example
%
%      n = -5:1:5;
%      a = mytf(n);
%      da_dn = mydtf(n,a);
%      subplot(2,1,1), plot(n,a)
%      subplot(2,1,2), plot(n,da_dn)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
% ** Replace the following calculation with your
% ** derivative calculation.
d = -8*n.^7.*a.^2;
% ** Note that you have both the transfer functions input N and
% ** output A available, which can often allow a more efficient
% ** calculation of the derivative than with just N.

```

可以将函数 `mydtf()` 作为一个模板，用来生成自定义的传递函数导数函数。

27.3.3 网络输入函数

网络输入函数根据给定的加权输入向量（或者矩阵） Z ，计算某层的网络输入向量（或矩阵） N ，网络输入向量和加权输入向量必须具有相同的维数。一旦定义了网络输入函数，就可以嵌入网络中任意一层。假设定义了网络输入函数为 `xiurf()`，并嵌入网络第三层，则应用下面语句实现：

```
net.layers{3}.netInputFcn='xiurf'
```

这样，在对网络进行仿真时，都可以应用此时设定的网络输入函数。

```
[Y, Pf, Af]=sim(net, P, Pi, Ai)
```

网络输入函数编制完成后，可以应用如下方式进行调用：

```
N=xiurf(Z1, Z2, ...)
```

其中， $Z1$ 、 $Z2$ 、 \dots 为加权输入向量； N 为函数返回值，即网络输入向量。

自定义网络输入函数返回如下信息。

- **version**：神经网络工具箱的版本。
- **deriv**：相关导数函数名。

神经网络工具箱中包含了一个自定义网络输入函数 `mynif()`，输入 `help mynif` 就可以获得有关此函数的帮助信息。举例说明如何应用 `mynif()` 进行网络输入向量计算。



【例 27-5】

```
Z1=rand(4,4);
Z2=rand(4,4);
Z3=rand(4,4);
N=mynif(Z1,Z2,Z3)
mynif('deriv')
```

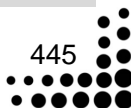
输出结果如下：

```
N =
    0.2289    0.0511    0.1041    0.1678
    0.1541    0.1653    0.0172    0.1194
    0.1605    0.1197    0.1231    0.0138
    0.1797    0.0064    0.1800    0.1630

ans =
deriv
```

输入 type mynif，可以查看 mynif()的源程序。

```
>> type mynif
function n=mynif(varargin)
%MYNIF Example custom net input function.
%
% Use this function as a template to write your own function.
%
% Calculation Syntax
%
% N = mynif(Z1,Z2,...)
% Zi - SxQ matrix of Q weighted (column) vectors.
% N - SxQ matrix of Q net input (column) vectors.
%
% Information Syntax
%
% info = mynif(code) returns useful information for each CODE string:
% 'version' - Returns the Neural Network Toolbox version (3.0).
% 'deriv' - Returns the name of the associated derivative function.
%
% Example
%
% z1 = rand(4,5);
% z2 = rand(4,5);
% z3 = rand(4,5);
% n = mynif(z1,z2,z3)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
```





```
if nargin < 1, error('Not enough arguments.');
```

```
end
n = varargin{1};
if isstr(n)
    switch n
        case 'version'
            a = 3.0;          % <-- Must be 3.0.
        case 'deriv'
            a = 'mydnif';    % <-- Replace with the name of your
                             %      associated derivative function or "
        otherwise
            error('Unrecognized code.')
```

```
end
else
% **  Replace the following calculation with your own.  The only
% **  constraint is that the function must not be sensitive
% **  to the order of its input arguments.
% **  In other words, MYNIF(Z1,Z2,Z3) must return the same
% **  values as MYNIF(Z2,Z3,Z1), MYNIF(Z1,Z3,Z2), etc.
    n = 1./n;
    for i=2:length(varargin)
        n = n + 1./varargin{i};
    end
    n = 1./n;
end
end
```

可以将函数 `mydnif()` 作为一个模板，用来生成自定义的网络输入函数。

27.3.4 网络输入导函数

如果自定义网络输入函数需要回传，则需要自定义网络输入导数函数，网络输入导数函数根据给定的加权输入来计算此层网络输入的导数。假设定义网络输入导数函数为 `xiurf()`，则应用下面语句计算此层网络输入的导数。

```
dN_dZ=xiurf(Z,N)
```

其中 Z 为网络加权输入向量； N 为网络输入向量； dN_dZ 表示导数 dN/dZ 。

神经网络工具箱中包含了一个自定义网络输入导数函数 `mydnif()`，输入 `help mydnif` 就可以获得有关此函数的帮助信息。下面举例说明如何应用 `mydnif()` 计算网络输入导数。

【例 27-6】

```
Z1=rand(4,4);
Z2=rand(4,4);
Z3=rand(4,4);
N=mydnif(Z1,Z2,Z3)
```



```
dN_dZ1=mydnif(Z1,N)
dN_dZ2=mydnif(Z2,N)
dN_dZ3=mydnif(Z3,N)
```

输出结果为:

```
N =
    0.1414    0.1639    0.1650    0.2149
    0.1784    0.0493    0.0874    0.1371
    0.0449    0.2010    0.1945    0.1432
    0.2311    0.0395    0.0145    0.1821
dN_dZ1 =
    0.0016    0.0091    0.0051    0.0130
    0.0070    0.0004    0.0004    0.0077
    0.0000    0.0107    0.0127    0.0009
    0.0522    0.0002    0.0001    0.0048
dN_dZ2 =
    0.0123    0.0170    0.0047    0.0272
    0.0147    0.0000    0.0007    0.0177
    0.0004    0.0147    0.0289    0.0201
    0.0172    0.0000    0.0000    0.0206
dN_dZ3 =
    0.0038    0.0028    0.0151    0.0216
    0.0079    0.0022    0.0005    0.0008
    0.0001    0.0213    0.0073    0.0144
    0.0221    0.0003    0.0002    0.0131
```

输入 type mydnif, 可以查看 mydnif() 的源程序。

```
>> type mydnif
function d = mydnif(z,n)
%MYDNIF Example custom net input derivative function of MYNIF.
%
%   Use this function as a template to write your own function.
%
%   Syntax
%
%       dN_dZ = dtansig(Z,N)
%       Z - SxQ matrix of Q weighted input (column) vectors.
%       N - SxQ matrix of Q net input (column) vectors.
%       dN_dZ - SxQ derivative dN/dZ.
%
%   Example
%
%       z1 = rand(4,5);
```





```
% z2 = rand(4,5);
% z3 = rand(4,5);
% n = mynif(z1,z2,z3)
% dn_dz1 = mydnif(z1,n)
% dn_dz2 = mydnif(z2,n)
% dn_dz3 = mydnif(z3,n)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.3.2.1 $
% ** Replace the following calculation with your
% ** derivative calculation.
d = n.^2 .* z.^2;
% ** Note that you have both the net input Z in question
% ** and output N available to calculate the derivative.
```

可以将函数 mydnif() 作为一个模板，用来生成自定义的网络输入导数函数。

27.3.5 权值函数

权值函数根据给定的输入向量（或者矩阵） P ，以及权值矩阵 W ，计算一个加权的输入向量（或矩阵） Z 。一旦定义了权值函数，就可以嵌入网络中任意输入权值和层权值上。假设定义权值函数为 xiurw()，并嵌入网络第一个输入到第三层的权值上，则应用下面语句实现。

```
net.inputWeights{3,1}.weightFcn='xiurw'
```

这样，在网络进行仿真时，都可以应用此时设定的权值函数。

```
[Y, Pf, Af]=sim(net, P, Pi, Ai)
```

权值函数编制完成后，可以应用如下方式进行调用：

```
Z=xiurw(W,P)
```

其中， P 为输入向量； W 为权值矩阵； Z 为加权输入向量。

自定义权值函数返回如下信息。

- **version:** 神经网络工具箱的版本。
- **deriv:** 相关导数函数名。

神经网络工具箱中包含了一个自定义权值函数 mywf()，输入 help mywf 就可以获得有关此函数的帮助信息。下面举例说明如何应用 mywf() 进行权值设置。

【例 27-7】

```
W=rand(1,4)
P=rand(4,1)
Z=mywf(W,P)
mywf('deriv')
```

输出结果如下：



```

W =
    0.1338    0.2071    0.6072    0.6299
P =
    0.3705
    0.5751
    0.4514
    0.0439
Z =
    0.2118

```

输入 `type mywf`，可以查看 `mywf()` 的源程序。

```

>> type mywf
function z=mywf(w,p)
%MYWF Example custom weight function.
%
% Use this function as a template to write your own function.
%
% Calculation Syntax
%
% Z = mywf(W,P)
%     W - SxR weight matrix.
%     P - RxQ matrix of Q input (column) vectors.
%     Z - SxQ matrix of Q weighted input (column) vectors.
%
% Information Syntax
%
% info = mytf(code) returns useful information for each CODE string:
%     'version' - Returns the Neural Network Toolbox version (3.0).
%     'deriv'   - Returns the name of the associated derivative function.
%
% Example
%
% w = rand(1,5);
% p = rand(5,1);
% z = mywf(w,p)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

if nargin < 1, error('Not enough arguments.');
```

```
end
if isstr(w)
    switch (w)
        case 'version'
            a = 3.0;          % <-- Must be 3.0.
```





```

case 'deriv'
    a = 'mydtf'; % <-- Replace with the name of your
                % associated function or "
otherwise
    error('Unrecognized code.')
end
else
% ** Replace the following calculation with your
% ** weighting calculation. The only constraint, if you
% ** want to define a derivative function, is that Z must
% ** be a sum of i terms, where each ith term is only a
% ** a function of w(i) and p(i).
    z = w*(p.^2);
end

```

可以将函数 `mywf()` 作为一个模板，用来生成自定义的权值函数。

27.3.6 权值导数函数

如果自定义权值函数需要回传，则需要自定义权值导数函数，权值导数函数根据输入和权值计算此层的加权输入的导数。假设定义权值导数函数为 `xiudrf()`，则应用下面语句计算此层加权输入的导数。

```

dZ_dP=xiudrf('p', W, P, Z)
dZ_dW=xiudrf('w', W, P, Z)

```

其中，`W` 为权值矩阵；`P` 为输入向量；`Z` 为加权输入向量；`dZ_dP` 表示导数 dZ/dP ；`dZ_dW` 表示导数 dZ/dW 。

神经网络工具箱中包含了一个自定义权值导数函数 `mydwf()`，输入 `help mydwf` 就可以获得有关此函数的帮助信息。下面举例说明如何应用 `mydwf()` 计算加权输入导数。

【例 27-8】

```

W=rand(1,4)
P=rand(4,1)
Z=mywf(W,P)
dZ_dP=mydwf('p',W,P,Z)
dZ_dW=mydwf('w',W,P,Z)

```

输出结果为：

```

W =
    0.7176    0.6927    0.0841    0.4544

P =
    0.4418
    0.3533

```



```

0.1536
0.6756
Z =
0.4359
dZ_dP =
0.6341    0.4894    0.0258    0.6140
dZ_dW =
0.1952
0.1248
0.0236
0.4565

```

输入 type mydwf, 可以查看 mydwf() 的源程序。

```

>> type mydwf
function d=mydwf(code,w,p,z)
%MYDWF Example custom weight derivative function for MYWF.
%
% Use this function as a template to write your own function.
%
% Syntax
%
% dZ_dP = mydwf('p',W,P,Z)
% dZ_dW = mydwf('w',W,P,Z)
% W - SxR weight matrix.
% P - RxQ matrix of Q input (column) vectors.
% Z - SxQ matrix of Q weighted input (column) vectors.
% dZ_dP - SxR derivative dZ/dP.
% dZ_dW - RxQ derivative dZ/dW.
%
% Example
%
% w = rand(1,5);
% p = rand(5,1);
% z = mywf(w,p)
% dz_dp = mydwf('p',w,p,z)
% dz_dw = mydwf('w',w,p,z)
%
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $
% ** Replace the following calculations with your
% ** derivative calculation. The only constraint is that
% ** the weight function must be a sum of elements, where
% ** each element i is a function of w(i) and p(i) only.

```





```

switch code
    case 'p', d = 2*w.*p;
    case 'w', d = p.^2;
    otherwise, error(['Unrecognized code.'])
end
% ** Note that you have both the transfer functions input N and
% ** output A available, which can often allow a more efficient
% ** calculation of the derivative than with just N.

```

可以将函数 `mydwf()` 作为一个模板，用来生成自定义的权值导数函数。

27.4 自组织函数

本节介绍如何自建自组织 SOM 网络中所用到的拓扑函数距离函数。

27.4.1 拓扑函数

在指定维数的情况下，拓扑函数可以计算某一层中神经元的位置。一旦定义了拓扑函数，就可以嵌入网络任意一层中。假设定义了拓扑函数为 `xiutop()`，并嵌入网络第三层，则应用下面语句实现。

```
net.layers{2}.topologyFcn='xiutop';
```

这样，任何时候网络进行训练或者自适应调节时，都可以应用此时设定的拓扑函数。

```

[net, tr]=train (NET, P, T, Pi, Ai)
[net, Y, E, Pf, Af]=adapt (NET, P, T, Pi, Ai)

```

拓扑函数编制完成后，输入一定的维数。调用拓扑函数，就可以计算出神经元的位置 `pos`，调用方式如下：

```
pos=xiutop (dim1, dim2, ... , dimN)
```

神经网络工具箱中包含一个自定义拓扑函数 `mytopf()`，输入 `help mytopf` 就可以获得有关此函数的帮助信息。下面举例说明如何应用 `mytopf()` 拓扑函数计算神经元的位置。

【例 27-9】 绘制应用 `mytopf()` 函数计算神经元位置的拓扑结构图，如图 27-3 所示。

```

pos=mytopf(15,15);
plotsom(pos)

```

输入 `type mytopf`，可以查看 `mytopf()` 的源程序。

```

>> type mytopf
function pos=mytopf(varargin)
%MYTOPF Example custom topology function.

```



```
%
% Use this function as a template to write your own function.
%
% Syntax
```

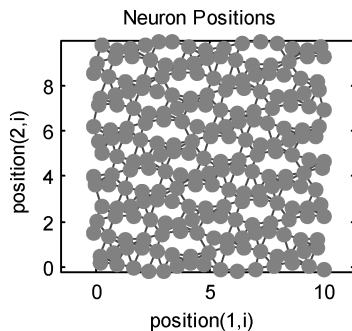


图 27-3 拓扑结构

```
% pos = mytopf(dim1,dim2,...,dimN)
%   dimi - number of neurons along the ith layer dimension
%   pos - NxS matrix of S position vectors, where S is the
%         total number of neurons which is defined by the
%         product dim1*dim1*...*dimN.
%
% Example
%
%   pos = mytopf(20,20);
%   plotsom(pos)

% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.2.2.1 $

% ** Replace the code below with your own calculation
% ** for the neuron positions.

dim = [varargin{:}]; % The dimensions as a row vector
size = prod(dim); % Total number of neurons
dims = length(dim); % Number of dimensions
pos = zeros(dims,size); % The size that POS will need to be
len = 1;
pos(1,1) = 0;
for i=1:length(dim)
    dimi = dim(i);
    newlen = len*dimi;
```



```

pos(1:(i-1),1:newlen) = pos(1:(i-1),rem(0:(newlen-1),len)+1);
posi = 0:(dimi-1);
pos(i,1:newlen) = posi(floor((0:(newlen-1))/len)+1);
len = newlen;
end
for i=1:2
    pos(i,:)=pos(i,:)*0.7+sin([1:size]*exp(1)/5*i)*0.2;
end

```

可以将函数 `mytopf()` 作为一个模板，用来生成自定义的拓扑函数。

27.4.2 距离函数

在指定位置的情况下，距离函数可以计算某一层中神经元的距离。一旦定义了距离函数，就可以嵌入网络任意一层中。假设定义了距离函数为 `xiudistf()`，并嵌入网络第三层，则应用下面语句实现。

```
net.layers{3}.distanceFcn='xiudistf';
```

这样，任何时候网络进行训练或者自适应调节时，都可以应用此时设定的距离函数。

```

[net, tr]=train(NET, P, T, Pi, Ai)
[net, Y, E, Pf, Af]=adapt(NET, P, T, Pi, Ai)

```

距离函数编制完成后，输入一定的位置信息。调用距离函数，就可以计算出距离，调用方式如下：

```
d=xiudistf(pos1, pos2, ..., posN)
```

神经网络工具箱中包含一个自定义距离函数 `mydistf()`，输入 `help mydistf` 就可以获得有关此函数的帮助信息。下面举例说明如何应用 `mydistf()` 距离函数计算神经元之间的距离。

【例 27-10】

```

pos=gridtop(2,3);
d=mydistf(pos)

```

输出结果为：

```

d =
    0    1.0000    1.0000    1.5874    2.0000    2.4473
    1.0000         0    1.5874    1.0000    2.4473    2.0000
    1.0000    1.5874         0    1.0000    1.0000    1.5874
    1.5874    1.0000    1.0000         0    1.5874    1.0000
    2.0000    2.4473    1.0000    1.5874         0    1.0000
    2.4473    2.0000    1.5874    1.0000    1.0000         0

```

输入 `type mydistf`，可以查看 `mydistf()` 的源程序。



```
>> type mydistf
function d = mydistf(pos)
%MYDISTF Example custom distance function.
%
%   Use this function as a template to write your own function.
%
%       Syntax
%
%       d = mydistf(pos)
%           pos - NxS matrix of S neuron position vectors.
%           d    - SxS matrix of neuron distances.
%
%   Example
%
%       pos = gridtop(3,2);
%       d = mydistf(pos)
% Copyright 1997 The MathWorks, Inc.
% $Revision: 1.3.2.1 $

s = size(pos,2);
for i=1:s
    for j=1:s
        % ** Replace the following line of code with your own
        % ** measure of distance.
        d(i,j) = norm(pos(:,i)-pos(:,j),1.5);
    end
end
end
```

可以将函数 mydistf() 作为一个模板，用来生成自定义的距离函数。

参考文献

- [1] 张良均, 曹晶, 蒋世忠. 神经网络的实用教程. 北京: 机械工业出版社, 2007.
- [2] 董长虹. MATLAB 神经网络与应用 (第 2 版). 北京: 国防工业出版社, 2006.
- [3] 周开利, 康耀红. 神经网络模型及其 MATLAB 仿真程序设计. 北京: 清华大学出版社, 2004.
- [4] 葛哲学. 精通 MATLAB. 北京: 电子工业出版社, 2008.
- [5] 闻新, 周露, 李翔, 等. MATLAB 神经网络仿真与应用. 北京: 北京科学出版, 2003.
- [6] 韩力群. 人工神经网络理论设计与应用. 北京: 化学工业出版社, 2002.
- [7] 飞思科技产品研发中心. 神经网络理论与 MATLAB 实现. 北京: 电子工业出版社, 2005.
- [8] 高隼. 人工神经网络原理及仿真实例 (第 2 版). 北京: 机械工业出版社, 2006.
- [9] 冯定. 神经网络专家系统. 北京: 科学出版社, 2006.
- [10] 丛爽. 面向 MATLAB 工具箱的神经网络理论与应用 (第 3 版). 北京: 中国科学技术大学出版社, 2008.
- [11] 田景文, 高美娟. 人工神经网络算法研究及应用. 北京: 北京理工大学出版社, 2006.
- [12] 韩力群. 人工神经网络教程. 北京: 北京邮电大学出版社, 2006.
- [13] 葛哲学, 孙志强. 神经网络理论与 MATLABR2007 实现. 北京: 电子工业出版社, 2007.
- [14] 庞中华, 崔红. 系统辨识与自适应控制 MATLAB 仿真. 北京: 北京航空航天大学出版社, 2009.
- [15] 于浩洋, 初红霞, 王希凤. MATLAB 实用教程——控制系统仿真与应用. 北京: 化学工业出版社, 2009.
- [16] 赵震宇, 徐用懋. 模块理论和神经网络的基础和应用. 北京: 清华大学出版社, 1996.
- [17] 薛定宇, 陈阳泉. 高等应用数学问题的 MATLAB 求解. 北京: 清华大学出版社, 2004.
- [18] 阎平凡, 张长水. 人工神经网络与模拟进货计算 (第 2 版). 北京: 清华大学出版社, 2005.
- [19] 石辛民, 郝整清. 模糊控制及其 MATLAB 仿真. 北京: 清华大学出版社及北京大学出版社, 2008.
- [20] 史峰, 王小川, 郁磊, 李洋. MATLAB 神经网络 30 个案例分析. 北京: 北京航空航天大学出版社. 2010.
- [21] 周品. MATLAB 神经网络设计与应用. 北京: 清华大学出版社. 2013.